НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-методичний комплекс
«Інститут післядипломної освіти»

Кафедра системного програмування і спеціалізованих комп'ютерних систем

«До захисту допущено»

Завідувач кафедри
_____ Віталій Романкевич
(підпис) (ініціали, прізвище)

"___" червня 2020___ р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 123 «Комп'ютерна інженерія»
(код і назва)

на тему: Корпоративні сховища даних на базі Oracle Data Integrator_____
_____
_____

Виконав (-ла): слухач (-ка) IVкурсу, групи ЗКІ-зп71___
(шифр групи)

Сетрина Вікторія Вікторівна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент каф. СПіСКС, к.т.н., доцент Клятченко Я.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю Клятченко Ярослав Михайлович _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент ст. викл. каф. ПМА Мальчиков В. В. _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань
Слухач _____
(підпис)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-методичний комплекс
«Інститут післядипломної освіти»

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Віталій Романкевич
(підпис)

"___" червня 2020___ р.

ЗАВДАННЯ
на дипломний проект слухачу

IVкурсу, групи ЗКІ-зп71 Сетрина Вікторія Вікторівна _____
(прізвище, ім'я, по батькові)

1. Тема проекту Корпоративні сховища даних на базі Oracle Data Integrator
_____
_____ ,
керівник проекту доцент каф. СПіСКС, к.т.н., доцент Клятченко Я.М. __ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом НМК „ІПО" КПІ імені Ігоря Сікорського від «__» _____ 2020___ р. №1130-с

2. Термін подання слухачем проекту __ червня 2020р. _____

3. Вихідні дані до проекту див. Технічне завдання _____
_____
_____
_____

4. Зміст пояснювальної записки _____
аналіз існуючих рішень та обґрунтування теми дипломного проекту розроблення програмного забезпечення _____
_____
_____

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

Схема архітектури реляційної OLAP Схема структурна_____
Схема компонентів часу виконання ODI Схема структурна_____
Алгоритм побудови завантаження даних Схема алгоритму _____
Схема алгоритму завантаження даних на DDS рівні Схема алгоритму____

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	Вивчення літератури за тематикою проекту	20.11.2019	
2	Розроблення ті узгодження технічного завдання	10.01.2020	
3	Аналіз існуючих рішень	22.01.2020	
4	Підготовка матеріалів першого розділу дипломного проекту	03.02.2020	
5	Розроблення програмного забезпечення	13.02.2020	
6	Відлагодження програмного продукту	25.02.2020	
7	Підготовка матеріалів другого розділу дипломного проекту	14.03.2020	
8	Розроблення програмної частини	28.03.2020	
9	Підготовка матеріалів третього розділу дипломного проекту	25.04.2020	
10	Підготовка матеріалів четвертого розділу дипломного проекту	12.05.2020	
11	Підготовка графічної частини дипломного проекту	23.05.2020	
12	Оформлення документації дипломного проекту	05.06.2020	

Слухач _____ _____
(підпис) (ініціали, прізвище)

Керівник проекту _____ _____
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (52 с., 13 рис. 3 табл., 2 додатки).

Об’єкт розробки – створення ЕТЛ процесу завантаження даних за типом slowly change dimension type 2 при побудові корпоративного сховища даних.

ЕТЛ процес дозволяє: здійснювати завантаження даних з реляційних СУБД до сховища даних. Це універсальних механізм, який включає в себе як первине так и інкрементальне наповнення сховища даних. В якості бази даних використовувалась Oracle.

В ході розробки:

- Розроблено алгоритм завантаження даних;
- сформульовані вимоги до первинного ті інкрементального завантаження незалежно від типів даних;
- розроблено алгоритм маркування записів (нові, змінені, удалені, повернуті записи) ;
- розроблена структура СУБД Oracle;
- розроблено алгоритм оптимізації та швидкодії ЕТЛ процесу;

Упровадження цієї системи в побудову корпоративного сховища даних, дасть змогу швидко ті оптимально побудувати сховище даних, забезпечити аналітиків та користувачів репортами для покращення умов ведення бізнесу.

Ключові слова:

ЕТЛ процес, slowly change dimension, Oracle, архітектура сховища, ODI Oracle Data Integrator, OLAP, case-інструменти.

ANNOTATION

Qualification work includes an explanatory note (52 p., 13 Fig. 3 tab., 2 annexes).

The development object is the creation of the ETL data loading process of the slowly change dimension type 2 type when building the corporate data warehouse.

The ETL process allows you to load data from a relational DBMS into a data warehouse. This is a universal mechanism that includes both primary and incremental filling of data storage. Oracle was used as a database.

During development:

– Developed data loading algorithm;

– formulated requirements for the primary incremental loading, regardless of data types;

– developed an algorithm for marking records (new, modified, deleted, returned records);

– The structure of the Oracle DBMS has been developed;

– an algorithm for optimization and speed of the ETL process has been developed;

The implementation of this system in the construction of a corporate data warehouse will quickly make it possible to optimally build a data warehouse, provide analysts and users with reports to improve business conditions.

Keywords:

ETL process, slowly change dimension, Oracle, storage architecture, ODI Oracle Data Integrator, OLAP, case tools.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кільк. акр.	№ примірн.	Прим.
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.467100.002 ТЗ	Корпоративні сховища	4		
			даних на базі Oracle			
			Data Integrator			
			Технічне завдання			
	А4	ІАЛЦ.467100.003 ТП	Корпоративні сховища	2		
			даних на базі Oracle			
			Data Integrator			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ.467100.004ПЗ	Корпоративні сховища	53		
			даних на базі Oracle			
			Data Integrator			
			Пояснювальна записка			
	А1	ІАЛЦ.467100.005Д1	Корпоративні сховища	1		
			даних на базі Oracle			
			Data Integrator			
			Схема архітектури			
			реляційної OLAP			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467100.001.ОА		
Розробив		Сетрина В.В.			Корпоративні сховища даних на базі Oracle Data Integrator Опис альбому	Літ.	Аркуш 6 / Аркушів 80
Перевірив		Клятченко Я.М.					
Н.контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського НМК «ІПО» Каф. СПіСКС Гр. ЗКІ-зп71	
Затвердив		Романкевич В. О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кільк. акр.	№ примірн.	Прим.
	А1	ІАЛЦ.467100.006 Д2	Корпоративні сховища	1		
			даних на базі Oracle			
			Data Integrator			
			Схема компонентів часу			
			виконання ODI			
	А1	ІАЛЦ.467100.007 Д3	Корпоративні сховища	1		
			даних на базі Oracle			
			Data Integrator			
			Алгоритм побудови			
			завантаження даних			
	А1	ІАЛЦ.467100.008 Д4	Корпоративні сховища	1		
			даних на базі Oracle			
			Data Integrator			
			Схема алгоритму			
			завантаження даних на			
			DDS рівні			
		Диск CD-ROM	Текст ПЗ. Тексти програм.	1		
			Графічний матеріал			
			даних			

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: «Корпоративні сховища даних на базі Oracle Data Integrator».

Галузь застосування: побудова корпоративних сховищ даних.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розробки є завдання на виконання роботи ступеня «бакалавр комп'ютерної інженерії», затверджене кафедрою системного програмування і спеціалізованих комп’ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3.МЕТА РОЗРОБКИ

Метою даного проекту є створення ЕТЛ процесу завантаження даних для побудови сховища даних.

4.ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5.ТЕХНІЧНІ ВИМОГИ

Вимоги до програмного продукту, що розробляється

- Універсальність підходу завантаження даних
- можливість запуску у будь-який момент часу;
- відповідність з моделю slowly change dimension type 2;
- історичність побудови сховища даних;
- можливість побудови ієрархічних репортів на основі сховища даних;
- аналітичні можливості.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.04.2020
2.	Розроблення та узгодження технічного завдання	30.04.2020
3.	Аналіз існуючих рішень	05.05.2020
4.	Підготовка матеріалів першого розділу дипломного проекту	10.05.2020
5.	Підготовка матеріалів другого розділу дипломного проекту	18.05.2020
6.	Підготовка графічної частини дипломного проекту	20.05.2020
7.	Оформлення документації дипломного проекту	25.05.2020
8.	Попередній огляд матеріалів диплому на кафедрі	30.05.2020

Пояснювальна записка
до дипломного проекту

на тему: Корпоративні сховища даних на базі Oracle Data Integrator_____
_____
_____

Київ – 2020 року

ЗМІСТ

					ИАЛЦ.467100.004 ПЗ		
Зм.	**Арк.**	**№ докум.**	**Підпис**	***Дата***			
Розробив		Сетрина В. В.			Засоби для побудови корпоративних сховищ даних Пояснювальна записка	**Літ.**	**Аркуш** 1 / **Аркушів** 53
		Клятченко Я.М.					
Н. контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського НМК «ІПО» Каф. СПіСКС Гр. ЗКІ-зп71	
Затвердив		Романкевич В.О.					

ТЕРМІНИ ТА ВИЗНАЧЕННЯ:

API –прикладний програмний інтерфейс (англ. application programming interface).

Drop table — команда SQL (Data Definition language).

ACID (Atomicity, Consistency, Isolation, and Durability) – атомарність, сталість, ізоляція і довготривалість даних.

EDW (Enterprise Data Warehouse) – корпоративне сховище даних.

ERP (Enterprise Resource Planning) – корпоративне планування ресурсів.

HOLAP-системи — це комбінований варіант зберігання даних, який використовує обидва типи СУБД. У багатовимірній СУБД зберігаються агрегати даних, а докладні дані, які мають невеликий обсяг, зберігаються в реляційній СУБД.

MOLAP - (Multidimensional OLAP), ROLAP (Relational OLAP) і HOLAP (Hybrid OLAP). В MOLAP-системі гіперкуб реалізується як спеціальна модель нереляційної структури, яка швидше забезпечує доступ до даних, ніж реляційні моделі, але вимагає додаткових витрат пам’яті.

OLTP (англ. *Online Transaction Processing*) — онлайнова обробка транзакцій. Спосіб організації БД, при якому система працює з невеликими за розмірами транзакціями, що йдуть великим потоком, і при цьому потрібний від системи максимально швидкий час відповіді.

MVC — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення (англ. Model-view-controller).

REST –Передача стану подання (англ. Representation State Transfer).

ROLAP — системах гіперкуб це лише користувацький інтерфейс, який моделюється на традиційній реляційній базі даних. Дані в сховищі представляються у вигляді моделі, що дістала назву «зірка» (star schema). Ця модель складається з таблиць двох типів: однієї таблиці даних, що аналізуються, тобто фактів (fact table) — центр зірки і декількох таблиць, які характеризують певні виміри цих фактів (demension table). Таблиця фактів

вміщує числові характеристики якогось напрямку діяльності компанії чи фірми, наприклад обсяги продажу, а також ключі таблиць вимірів. Таблиці вимірів містять додаткові характеристики ключових полів, як правило, це довідкові дані, наприклад дані про назву товару, назву його виробника, тип товару та інші. Зауважимо, що дані таблиць вимірів денормалізовані.

SQL (мова структурованих запитів) —декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних і її модифікації, системи контролю за доступом до бази даних (англ. Structured query language).

БД –база даних (англ. database).

ПЗ –програмне забезпечення.

ВСТУП

Компанії оперують великим обсягом інформації, тому стикаються з проблемою якості даних і відсутністю даних про минулі періоди в потрібних розрізах. Інформація із зовнішніх і внутрішніх джерел має різну структуру і дані часто дублюються або суперечать один одному. Корпоративні сховища даних вирішують ці проблеми.

Корпоративні сховища даних накопичують інформацію в єдиному джерелі про бізнес компанії для зовнішніх і внутрішніх користувачів: керівництва, співробітників і клієнтів.

Спеціально розробляються рішення, які централізують великий обсяг інформації і підвищують якість даних для прийняття управлінських рішень.

Отже, основним завданням будь-якої корпоративної інформаційної системи є оперативне забезпечення достовірною інформацією управлінського персоналу для прийняття оптимальних рішень. Це завдання значно полегшується, якщо КІС базується на використанні сховища даних. Корпоративне сховище даних є серцевиною сучасних КІС, і від того, як воно побудоване, залежить ефективність роботи всієї системи.

Корпоративне сховище даних – це спеціальним чином організований масив даних підприємства (організації), що обробляється і зберігається в єдиному апаратно-програмному комплексі, який забезпечує швидкий доступ до оперативної та історичної інформації, багатовимірний аналіз даних (KPI по різним вимірам), отримання прогнозів і статистики в розрізах узгодженої нормативно-довідкової інформації (НДІ).

У сховищі акумулюється інформація з різних джерел, що характеризують бізнесову діяльність корпорації, яка зберігається у вигляді, що забезпечує практичні потреби користувача. Оперативне аналітичне оброблення даних за допомогою технології OLAP дозволяє аналітикам, менеджерам і управлінцям сформувати власне бачення даних, використовуючи швидкий, одноманітний, оперативний доступ до різноманітних форм подання інформації.

РОЗДІЛ 1
ЗАГАЛЬНА ХАРАКТЕРИСТИКА ПОБУДОВИ КОРПОРАТИВНИХ СХОВИЩ ДАНИХ

1.1 Сутність корпоративних сховищ даних

Спроби створення систем прийняття рішень, які б безпосередньо зверталися до баз даних систем оперативної обробки трансакцій (OLTP-систем), виявляються в більшості випадків не ефективними. Тому для забезпечення можливості аналізу накопичених даних організації почали створювати сховища даних (***Data Warehouse*** – DW), що являють собою інтегровані колекції даних, зібрані з різних систем оперативного доступу до даних.

Концепція DW була запропонована в 1992 р. Білом Інмоном в його книзі “Building the Data Warehouse” та стала однією з домінуючих в розробці інформаційних технологій обробки даних 90-х років. Англомовний термін Data Warehousing, який складно лаконічно перекласти українською, означає створення, підтримку, управління та використання сховища даних, що говорить про те, що мова йде про процес. Мета цього процесу – неперервне надання необхідної інформації потрібним співробітникам організації. Цей процес передбачає постійний розвиток, удосконалення, розв’язання все нових задач. Процес ніколи не закінчується, тому його не можна вмістити в більш-менш чіткі часові рамки так, як це можна зробити для традиційних систем оперативного доступу до даних.

Сховища даних є основою для побудови систем підтримки прийняття рішень. Основна мета створення DW в тому, щоб зробити усі значимі для управління бізнесом дані доступними в стандартизованій формі, придатними для аналізу та отримання необхідних звітів. Для досягнення цього потрібно отримати дані із існуючих внутрішніх та зовнішніх, доступних для комп’ютера, джерел. Незважаючи на відмінності в підходах та реалізаціях,

усім сховищам даних властиві такі спільні риси [Error: Reference source not found]: ***предметна орієнтованість***, ***інтегрованість***, ***прив'язка до часу***, ***незмінність***.

Предметна орієнтованість. Інформація в сховищі даних організована у відповідності до основних аспектів діяльності підприємства (замовники, продажі, склад тощо). Це відрізняє сховище даних від оперативної БД, де дані організовано відповідно до процесів (виписка рахунків, відвантаження товару тощо). Предметна організація даних в сховищі сприяє як значному спрощенню аналізу, так і підвищенню швидкості виконання аналітичних запитів. Вона виражається, зокрема, в використанні інших, порівняно з оперативними системами, систем організації даних. У випадку зберігання даних в реляційній СУБД використовується схема „зірки" (star) чи „сніжинки" (snowflake) [Error: Reference source not found]. Крім цього, дані можуть зберігатися в спеціальній багатовимірній СУБД в n-вимірних кубах.

Інтегрованість. Вихідні дані отримуються із оперативних БД, перевіряються, очищуються, приводяться до єдиного виду, в потрібній мірі агрегуються (вираховуються сумарні та інші статистичні показники) і завантажуються в сховище. Такі інтегровані дані набагато простіше аналізувати.

Прив'язка до часу. Дані в сховищі завжди напряму зв'язані з певним періодом часу. Дані, отримані із оперативних БД, накопичуються в сховищі у виді „історичних шарів", кожен з яких стосується конкретного періоду часу. Це дозволяє аналізувати тенденції в розвитку бізнесу.

Незмінність. Потрапивши в певний „історичний шар" сховища, дані уже ніколи не мінятимуться. Це також відрізняє сховище від оперативної БД, в якій дані постійно змінюються, у зв'язку з чим один і той же запит, виконаний в різні моменти часу, може дати різні результати. Стабільність даних також полегшує їх аналіз.

Сховища даних умовно поділяють на два типи [Error: Reference source not found]: ***корпоративні сховища даних*** (enterprise data warehouses) та ***кіоски даних*** (data marts). Корпоративні сховища даних містять інформацію, яка стосується усієї корпорації (всього підприємства), і яка

зібрана з великої кількості оперативних джерел для консолідованого аналізу. Зазвичай такі сховища охоплюють цілий ряд аспектів діяльності підприємства і використовуються для прийняття як тактичних, так і стратегічних рішень. Корпоративне сховище містить детальну та узагальнюючу інформацію. Вартість створення та підтримки корпоративних сховищ може бути дуже великою. Частіше всього їх створенням займаються централізовані відділи інформаційних технологій, причому вони створюються методом ***зверху вниз*** – спочатку проектується загальна схема, і тільки потім починається заповнення даними. Такий процес може тривати декілька років.

Кіоски даних містять підмножину корпоративних даних та створюються для відділів чи підрозділів всередині організації. Кіоски даних часто створюються силами самого відділу та охоплюють конкретний аспект, що цікавить співробітників даного відділу. Кіоск даних може отримувати дані з корпоративного сховища (***залежний кіоск***) або, що більш розповсюджено, дані можуть отримуватись безпосередньо з оперативних джерел (***незалежний кіоск***).

Основними постачальниками програмного забезпечення сховищ даних є компанії Arbor, Hewlett-Packard, IBM, Informix, Microsoft, Oracle, Platinum Technology, SAS Institute, Software AG, Sybase та ін. Усі ці фірми мають сторінки в Internet, на яких наводяться детальні відомості про їх продукти та послуги.

1.2 Методи побудови сховищ даних

Враховуючи специфіку, сховище даних має такі особливості проектування та побудови:

1) отримання інформації з різних джерел даних (у тому числі і з реляційних баз даних) у деталізованому та аґреґованому вигляді (зберігаються результати застосування функцій агрегації – суми, середнього значння, максимуму, мінімуму тощо);

2) багатовимірне подання інформації – ігноруються деякі вимоги нормалізації (дотримують максимум 3-ої нормальної форми), що значно

підвищує швидкість опрацювання інформації, оскільки зменшує кількість операцій з'єднання;

3) наявність метаданих для опису джерел метаданих та структури самого сховища даних – у базах даних також використовують словники для опису структур даних, а у сховищах даних мета дані (словники, дані про дані) повинні будуватися за класифікаційною схемою Захмана. За цією схемою описую об'єкти (що?), суб'єкти (хто?), місцезнаходження (де?), час (коли?), фактори впливу, чинники (чому?), способи (як?);

4) наявність пакетного завантаження даних в сховище даних та вивантаження даних;

5) наявність процедур аналізу даних та отримання нових даних;

6) орієнтованість даних на аналітичне, а не на статичне опрацювання.

Сховища даних краще пристосовані до зберігання та аналітичного опрацювання великих обсягів даних і, в основному, є інтеґрацією реляційної та багатовимірної моделей. На сьогодні є такі архітектури побудови сховищ даних: *корпоративна інформаційна фабрика Білла Інмона, шина Ральфа Кімбола, зведення даних корпорації TDAN*. Вони мають розвинені засоби інтеґрації даних з різних джерел та дозволяють працювати як з деталізованою, так і аґреґованою інформацією.

Парадигма для реляційних даних в сховищі даних (парадигма *корпоративної інформаційної фабрики* КІФ – Corporate Information Factory, CIF) розроблена Інмоном і передбачає, що дані повинні перебувати на низькому рівні ступені деталізації і в третій нормальній формі (3НФ, 3NF). Білл Інмон підтримує повторний або спіральний підхід до розвитку великого сховища даних. За цим підходом розвиток сховища відбувається ітераційно, тобто у разі виникнення потреби додається одна таблиця за один раз, що забезпечує лише незначну зміну схеми даних. Тому такий підхід до проектування сховища ще називають *спіральним підходом*.

1.3 Архітектура сховищ даних

Основним завданням будь-якої корпоративної інформаційної системи є оперативне забезпечення достовірною інформацією управлінського персоналу для прийняття оптимальних рішень. Це завдання значно

полегшується, якщо КІС базується на використанні сховища даних. Корпоративне сховище даних є серцевиною сучасних КІС, і від того, як воно побудоване, залежить ефективність роботи всієї системи.

У сховищі акумулюється інформація з різних джерел, що характеризують бізнесову діяльність корпорації, яка зберігається у вигляді, що забезпечує практичні потреби користувача. Оперативне аналітичне оброблення даних за допомогою технології OLAP дозволяє аналітикам, менеджерам і управлінцям сформувати власне бачення даних, використовуючи швидкий, одноманітний, оперативний доступ до різноманітних форм подання інформації.

Перш ніж перейти до характеристики архітектурних рівнів сховища даних, спинімося на технології комплексного багатовимірного аналізу даних, що дістала назву OLAP (On-Line Analytical Processing). OLAP — це головний компонент організації сховищ даних (Data Warehousing), тобто збирання, очищення й попереднього оброблення даних з метою надання результативної інформації користувачам для оперативного аналізу та складання звітів. Концепцію OLAP сформулював і описав 1993 року Е. Ф. Кодд, відомий дослідник баз даних і автор реляційної моделі даних.

У 1993 році Е. Ф. Кодд із партнерами опублікували статтю «Забезпечення OLAP для користувачів-аналітиків», де описали 12 правил OLAP. Пізніше (1995-го) до них було долучено іще шість правил. Ці правила Е. Ф. Кодд розбив об’єднав у групи, назвавши їх особливостями, які ми розглянемо далі.

1. Багатовимірне концептуальне подання даних (правило 1). Це оригінальне правило є серцевиною OLAP.

2. Інтуїтивне маніпулювання даними (правило 10). Це означає, що маніпулювання даними має здійснюватися безпосередніми діями над комірками в режимі перегляду без застосування меню і численних операцій. Нині таким інструментом є використання мишки чи інших подібних пристроїв.

3. Доступність (правило 3). У цьому правилі Е. Ф. Кодд особливо наголошує роль OLAP як посередника (шару) між гетерогенними джерелами даних і

поданням їх кінцевому користувачеві.

4. Пакетне вилучення замість інтерпретації (нове правило із 1995 року). Це правило вимагає, щоби програмний продукт однаковою мірою ефективно забезпечував доступ як до власного сховища даних, так і до зовнішніх даних. Зауважимо, що Е. Ф. Кодд наполягав на багатовимірному поданні даних із частковими попередніми обчисленнями для великих багатовимірних баз даних у такий спосіб, щоб будь-які детальні дані були прозорі й доступні. Сьогодні цьому визначенню відповідають гібридні OLAP, які стають найпопулярнішими, що підтверджує далекоглядність Е. Ф. Кодда.

5. Моделі аналізу OLAP (нове правило). Згідно з цим правилом доктор Кодд вимагає, щоб OLAP-продукти підтримували всі чотири моделі аналізу, які він описує у своїй статті (категоріальний, тлумачний, абстрактний і стереотипний). Ідеться, відповідно, про формування параметрично налаштовуваних звітів, розрізів і групувань з обертанням, аналізом у стилі «що, якщо» і моделями пошуку цілей. Як правило, всі сучасні OLAP інструментальні засоби підтримують перші дві моделі аналізу, більшість із них — третю (певною мірою) і лише деякі — четверту модель в окремих корисних розширеннях. Можливо, у цьому правилі Е. Ф. Кодд передбачав системи добування даних (data mining), які ще називають моделями пошуку цілей.

6. Архітектура «клієнт-сервер» (правило 5). Це правило вимагає, щоб програмний продукт був не лише клієнт-серверним, а й щоб серверний був достатньо інтелектуальним аби різні користувачі могли підключатися з мінімумом зусиль і навичок програмування. Це правило диктує жорстку архітектуру. Але в теперішній час клієнт-серверна архітектура поступово витісняється Web-серверною, про яку Е. Ф. Кодд не згадував.

7. Прозорість (правило 2). Цілковита відповідність цьому правилу означає, що користувач електронної таблиці здатен отримувати всі необхідні дані з OLAP-системи, не підозрюючи навіть, звідки вони беруться. Щоб виконувати це правило, програмний продукт має забезпечувати безпосередній доступ до гетерогенних джерел даних і одночасно мати вбудовану повнофункціональну електронну таблицю. Це правило висуває

жорсткі вимоги до OLAP-систем, які в більшості продуктів не мають цілковитої підтримки, тобто не підтримують безпосереднього прозорого доступу до зовнішніх даних. Більшість програмних продуктів не надають або доступу до електронних таблиць, або безпосереднього доступу до гетерогенних джерел.

8. Багатокористувацька підтримка (правило 8). Це правило вказує на стратегічну спрямованість розвитку OLAP. Інструменти OLAP мають забезпечити одночасне читання і запис даних, їх інтеграцію та конфіденційність. Але й досі багато прикладень OLAP працюють лише в режимі читання.

9. Оброблення ненормалізованих даних (нове правило). Правило вказує на необхідність інтеграції між OLAP-системою і ненормалізованими джерелами даних. Згідно з цим правилом Е. Ф. Кодда модифікації даних, виконувані в середовищі OLAP, не мають приводити до змін даних, що зберігаються у вихідних зовнішніх системах.

10. Збереження результатів OLAP: зберігання їх окремо від вихідних даних (нове правило). У цьому правилі Е. Ф. Кодд дотримується поширеної думки про те, що OLAP-додатки, що працюють в режимі читання-запису, не мають безпосередньо впливати на оброблювані дані, а дані, модифіковані в OLAP, мають зберігатися окремо від даних транзакцій. Удалою реалізацією цього правила є метод зворотного запису даних, що використовується в Microsoft OLAP Services, який дозволяє зберігати дані, змінені в середовищі OLAP, окремо від основних даних.

11. Вилучення значень, яких немає (нове правило). Усі значення, яких немає, вилучаються з визначеної версією 2 реляційної моделі даних, тобто значення, яких немає, повинні відрізнятися від нульових. Це правило має рацію лише в плані компактності зберігання даних, утім деякі сучасні OLAP-інструменти ігнорують це правило без особливих втрат у функціональності.

12. Оброблення значень, яких немає (нове правило). Усі значення, яких немає, будуть ігноруватися OLAP-аналізатором без урахування їхнього джерела. Ця особливість пов’язана з попереднім правилом і є природним наслідком того, як OLAP-система обробляє всі дані.

Третя група правил (особливості подання звітів)

13. Гнучкість формування звітів (правило 11). Правило вимагає, щоб показники могли бути розміщені у звіті так, як це потрібно користувачу. Більшість сучасних програмних продуктів можуть це забезпечити у своїх засобах формування звітів, але не всі здатні забезпечити цю гнучкість в інтерактивному режимі під час перегляду. Для цього необхідно, щоб засоби аналізу й можливості формування звітів були комбіновані в одному програмному модулі.

14. Стандартна продуктивність звітів (правило 4). У цьому правилі Е. Ф. Кодд вимагає, щоби продуктивність формування звітів істотно не зменшувалася зі зростанням кількості вимірів і розмірів бази даних. Зауважимо, що з приводу цієї особливості між програмними продуктами існують серйозні відмінності, однак принциповим чинником впливу на продуктивність є кількість виконуваних обчислень і те, де вони виконуються (на машині клієнта, на сервері багатовимірної бази даних чи в реляційній СУБД). Досвід показує, що просте збільшення кількості вимірів або розмірів БД істотно не впливає на продуктивність у базах даних із повним попереднім обчисленням, що й мав на увазі Е. Ф. Кодд у своєму твердженні. Але звіти з великою місткістю обчислювальних показників формуються довше. Продуктивність практично лінійно залежить від кількості використовуваних під час формування звіту комірок, яких може бути значно більше, ніж тих, що з’являються в кінцевому звіті. Деякі розміщення вимірів у звіті можуть бути повільнішими за інші, оскільки потребують більшої кількості операцій зчитування з диска.

15. Aklвтоматичне налаштування фізичного рівня (правило 7). Правило вимагає, щоб OLAP-система автоматично налаштовувала свою фізичну схему залежно від типу моделі, обсягів даних і розрядності бази даних. Ніхто не заперечує це твердження, але більшість постачальників OLAP-систем надто далекі від цього ідеалу.

16. Універсальність вимірів (правило 6). Е. Ф. Кодд наполягає на тому, що всі виміри мають бути рівноправними, кожнн вимір повинен бути еквівалентним і в структурі, і в операційних можливостях. Щоправда, він не заперечує

проти додаткових операційних можливостей для окремих вимірів (як правило, часових), але наполягає, щоб такі додаткові функції могли бути надані будьякому виміру. Дослідник також категорично виступає проти того, щоби базові структури даних, обчислювальні або звітні формати були більшою мірою притаманні якомусь одному виміру. Досвід експлуатації OLAP-систем довів, що це найсуперечливіше правило з усіх 12-ти початкових правил. Якщо технологія орієнтує програмний продукт на відповідність цьому правилу, то постачальники відповідних продуктів підтримують його. Якщо прикладне застосування зорієнтовує продукт не докладати зусиль на його дотримання, то постачальники програм ігнорують його. На практиці вироблено такий стереотип. Якщо OLAPсистема потрібна для універсальних цілей, для множинних досліджень, то це правило треба враховувати, а якщо продукт передбачено для специфічного використання, його можна ігнорувати.

17. Необмежена кількість вимірів і рівнів агрегації (правило 12). Слід сказати, що технічно неможливо реалізувати цю вимогу в повному розумінні необмеженості, оскільки не може бути необмеженого об'єкта на обмеженому комп'ютері. Принаймні небагато прикладних задач потребують більш як 8—10 вимірів.

18. Необмежені операції між розмірностями (правило 9). Згідно із твердженням Е. Ф. Кодда всі види операцій мають бути дозволені для будь-яких вимірів, а не лише для вимірів типу «міра». Більшість сучасних продуктів із багатовимірною базою даних задовольняють цю вимогу. Але програмні продукти, що використовують реляційну структуру зберігання даних, слабкі в цій галузі. Цей тип операцій важливий, якщо OLAP потрібна для комплексних складних досліджень, а не лише для перехресної вибірки даних.

Названі вище правила доцільно використовувати при визначенні того, який програмний продукт правомірно відносити до категорії OLAP. Але пам'ятати й використовувати 18 особливостей OLAP надто обтяжливо для більшості фахівців. Практика підказала, що можна описати OLAP-визначення п'ятьма головними словами — швидкий аналіз розподіленої багатовимірної

інформації. Техніка реалізації OLAP охоплює кілька архітектур зберігання даних для цих цілей. На рис. 1 показано структуру корпоративного сховища даних. Можна бачити, первинні дані можна завантажити в спеціальну базу даних OLAP або залишити в реляційній чи змішаній базі.

Поряд із потоками даних існують потоки метаданих, які забезпечують взаємодію різних компонентів сховища й розміщуються в репозитарії. Репозитарій є центральною частиною сховища даних. Репозитарій метаданих дає змогу визначити семантичну структуру додатка у вигляді опису термінів предметної галузі, їх взаємозв'язки й атрибути. Терміни предметної галузі, визначеної в репозитарії метаданих, можуть бути використані для створення розділюваних розмірностей і структури довідників для визначення взаємодії між документами.

Крім того, завданням метаданих є відстежування змін у структурі моделі предметної галузі та забезпечення порівнювання даних, зібраних у різні періоди. Властивість метаданих відстежувати зміни структур даних і їх значення в часі називається контролем модифікацій (versioning).

Корпоративне сховище даних може функціонувати у трьох архітектурах — реляційній (ROLAP), багатовимірній (MOLAP) і гібридній, або змішаній (HOLAP).

У ROLAP-архітектурі (Relational OLAP) детальні дані перебувають у реляційній БД, агреговані дані (підсумкові) — там само у спеціально створених службових таблицях. Реляційні таблиці та зв'язки між ними генеруються автоматично. Частина таблиць створюється під час інсталяції системи (таблиці для зберігання метаданих, таблиці обов'язкових реквізитів об'єктів, таблиці для зберігання залишків на рахунках бухгалтерського обліку), частина — під час опису логічної моделі даних (метаданих системи).

Підхід побудови ROLAP-архітектури (оперативного аналітичного оброблення реляційних даних) базується на посиланні, що дані не обов'язково мають зберігатися в багатовимірному вигляді для того, щоб потім їх можна було використати в багатовимірному аналізі. Виробники ROLAP, як правило, розподіляють головні функції системи OLAP між трьома логічними рівнями:

— масштабована паралельна реляційна база даних забезпечує зберігання і швидкий доступ до даних;

— середній рівень аналізу підтримує багатовимірне подання даних і розширені функціональні можливості, які недоступні на базовому реляційному сервері;

— рівень подання відповідає за донесення результатів до користувачів.

Переваги системи ROLAP полягають у повноті функціональних можливостей поряд з відкритістю, масштабованістю та продуктивністю, які є головними якостями реляційних баз даних провідних розробників.

На рис. 1.1 подано схему роботи зі сховищем даних у ROLAP-архітектурі. За такої архітектури на сервері створюється реляційне сховище, а на клієнтських машинах встановлюються інструментальні засоби оброблення запитів або механізм OLAP чи інші зовнішні системи.

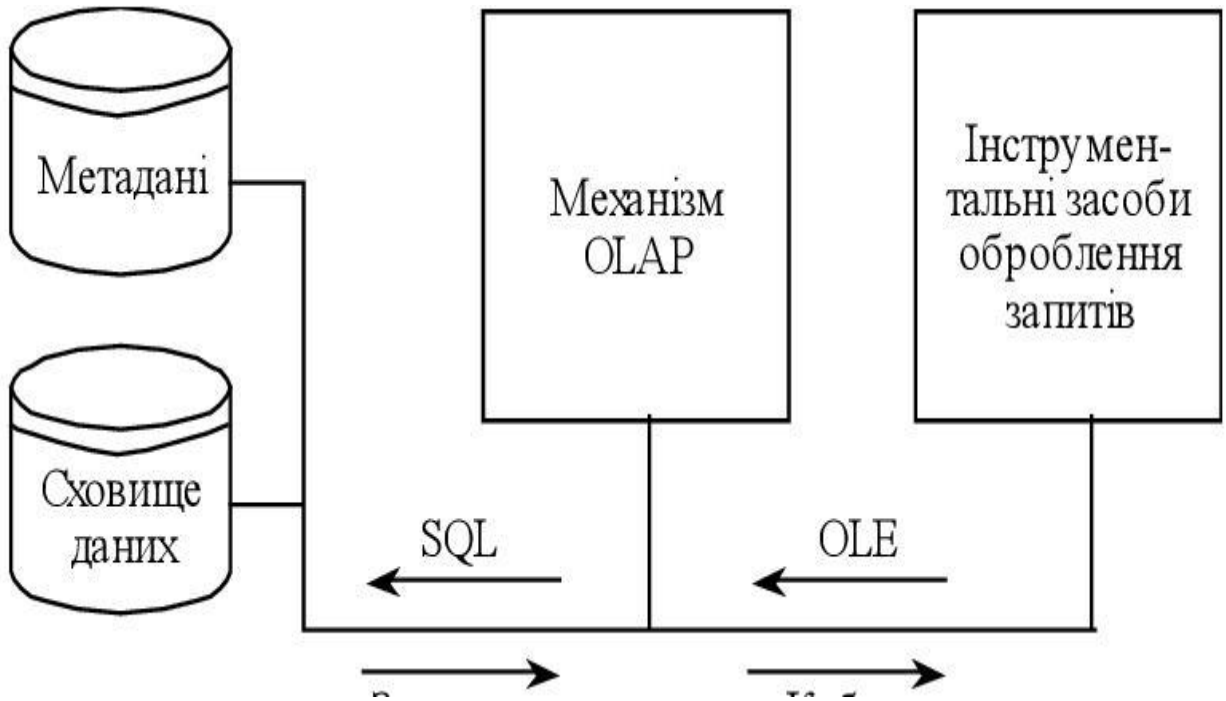

Рис.1.1 - Архітектура реляційної OLAP

Реляційна архітектура забезпечує високу швидкість роботи зі сховищем, в якому зберігається значний обсяг інформації. За дуже великих обсягів даних продуктивність системи, побудованої на ROLAP-архітектурі, значно знижується. У цьому разі для забезпечення високої швидкості багатовимірного аналізу будують гібридну OLAP-архітектуру (HOLAP).

HOLAP-архітектура (Hybrid OLAP) — це спеціалізований механізм, який дає змогу зберігати дані у власних форматах, які являють собою масиви, що відповідають зручному для користувачів поданні даних у так званих ділових вимірах. Основною ознакою цієї архітектури є те, що детальні дані залишаються на відведеному для них місці — в реляційному сховищі, а агреговані (підсумкові) дані зберігаються в багатовимірній базі (Multidimensional OLAP— MOLAP).

Висока швидкість оброблення запитів у багатовимірній базі даних забезпечується ефективним механізмом попереднього обчислення показників для задоволення запитів. Швидкість оброблення запитів значно підвищується за рахунок того, що можна отримати відповідь на запитання на підставі результатів попередніх обчислень, а не виконуючи їх «на льоту». Використання HOLAP-архітектури найефективніше в разі оброблення надто великих обсягів даних.

На рис. 1.2 зображено архітектуру гібридної OLAP.

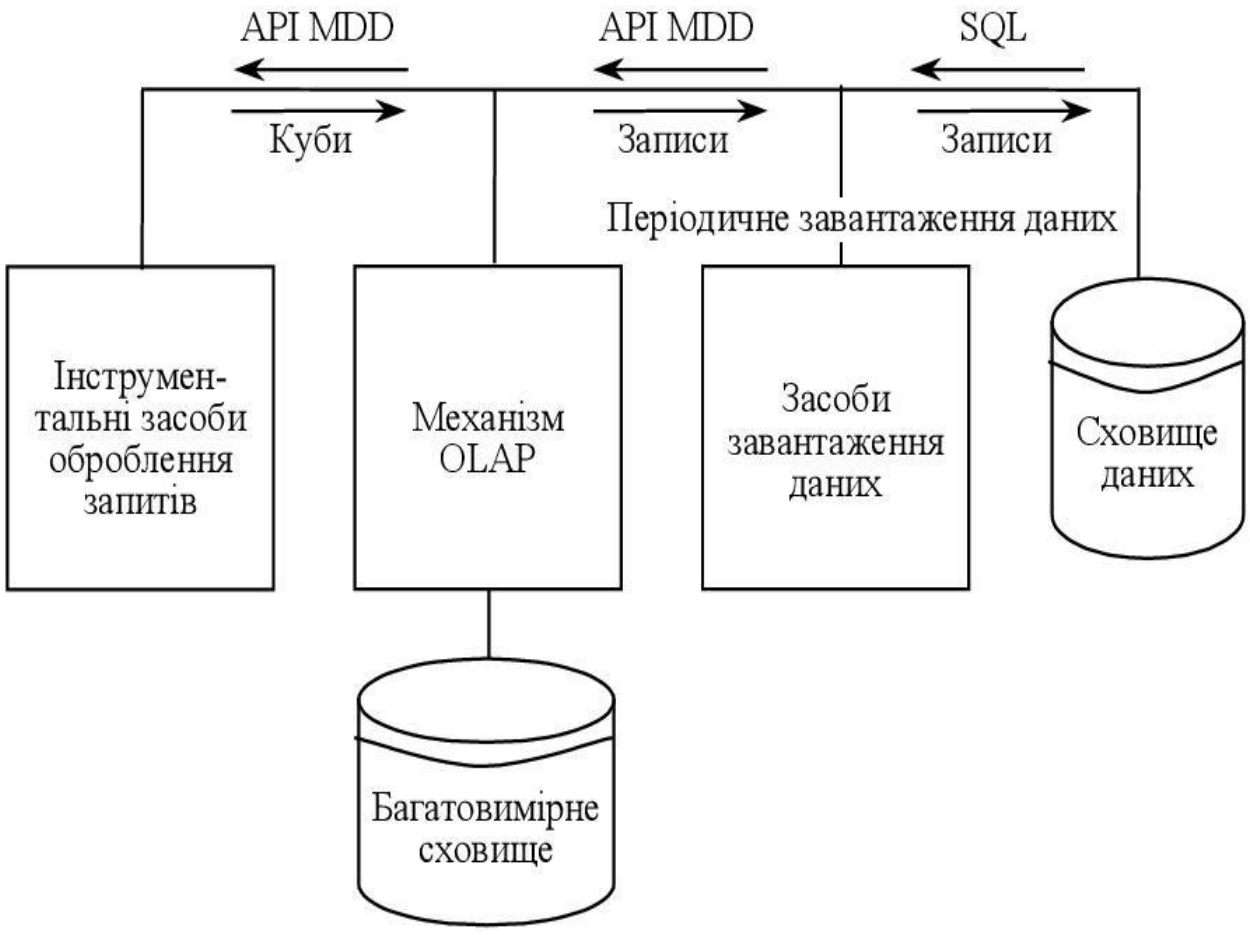

Рис.1.2 - Архітектура гібридної OLAP

Згідно з рис. 1.2 у реляційному сховищі збираються первинні дані корпорації. OLAP-механізм реалізований на базі багатовимірної СУБД. Як багатовимірне сховище можуть використовуватися вітрини даних. У вітрини даних зі сховища буде періодично імпортуватися агрегована інформація з вузьких предметних галузей. Створювати вітрини даних і підключати їх до корпоративного сховища можна, наприклад, за допомогою сервісу MS Analysis Services, який входить до складу MS SQL Server, або за допомогою подібних OLAP-серверів інших виробників. У цій архітектурі OLAP-клієнт працюватиме не безпосередньо з реляційним сховищем, а з багатовимірною БД (MOLAP). Щоб забезпечити актуальний OLAP-аналіз, необхідно регулярно поповнювати вітрини даними сховища.

Кожна з цих архітектур має певні переваги й вади та має використовуватися залежно від наявних умов — обсягу даних, потужності реляційної СУБД, парку ЕОМ тощо.

Нещодавно OLAP-продукти підтримували або реляційне, або багатовимірне зберігання даних. На теперішній час, як правило, той самий продукт забезпечує обидва ці види зберігання даних.

Ми розглянули архітектури корпоративних сховищ даних у плані зберігання інформації у сховищі. Але мову можна вести про побудову функціональної архітектури корпоративного сховища даних. Функціональна архітектура в кожному конкретному випадку матиме свої особливості та залежатиме від діяльності фірми, від її бізнес-платформи, територіального розподілу тощо.

Як приклад наведемо загальну функціональну архітектуру корпоративного сховища даних з огляду на те, що в його складі мають бути такі блоки:

— сховище даних. Структура сховища має бути зорієнтована на зберігання бізнес-даних корпорації;

— клієнтська частина системи. Клієнтська частина може охоплювати різноманітні програмні засоби залежно від потреб користувача. Як приклад наведемо програми, що містять дизайнер сховища, засоби розроблення додатків, засоби адміністрування користувачів, інструменти аналізу даних,

завантаження словника метаданих з XML-файла (eXtensible Markup Language) у сховище і вивантаження його зі сховища в XML-файл. Крім клієнтів системи можуть бути використані зовнішні OLAP-клієнти для аналізу даних;

— сервер обміну даними (Data Exchange Server). Це набір програм завантаження/вивантаження даних сховища й каталогів для організації обміну даними із зовнішніми OLTP-системами. Сервер забезпечує завантаження даних із XML-файлів відповідних форматів у сховище і вивантаження зі сховища в XML-файл;

— бібліотеки прикладних класів. Окрім загальновідомих бібліотек API (Application Program Interface), які вбудовуються в ядра операційних систем для абстрагування прикладних програм від типів устаткування і низькорівневих протоколів обміну інформацією, використовуються додаткові бібліотеки, що постачаються з багатьма засобами оброблення з метою зменшення трудомісткості й термінів розроблення програм. Найпоширеніші бібліотеки прикладних класів такі: ACL (Application Class Library), VCL (Visual Components Library), Win Lite тощо. ACL — це об’єктна оболонка сховища даних, яка вміщує структуру реляційних таблиць і процедур SQL, що зберігаються. Вона реалізована мовою Python і використовується для оброблення XML-документів та інших функцій. Кожен клас забезпечує інтерфейс для окремого об’єкта між його XML-поданням і поданням об’єкта в БД.

У Delphi використовується дуже потужна і складна бібліотека VCL (Visual Components Library), яка окрім безпосередніх абстракцій уводить також велику кількість своїх функціональних класів. У цій бібліотеці є компоненти для візуального відображення інформації, роботи з базами даних, із системними об’єктами, компоненти для роботи з Internet тощо. Win Lite — це компактна бібліотека оконних класів. Вона є мінімальною за розміром і не містить вищих рівнів абстракції, ніж існують у Win32 API, але значно полегшує роботу під час переведення програмування в об’єктно-

орієнтоване русло. Вона може бути використана разом з VCL-бібліотекою.

1.3.2 Підхід Inmon vs Kimpball

Перше істотна відмінність між цими архітектурами – різні підходи до побудови баз даних, що становлять основу Сховища. Якщо Ральф Кімболл (Ralph Kimball) використовує просторову організацію баз даних (dimensional data bases) с так званої архітектурою "зірка" як на стадії підготовки, так і презентації даних, то Білл Інмон (Bill Inmon) комбінує два підходи. У його моделі атомарні дані організовані в реляційні бази і знаходяться в нормалізованому Сховище даних, причому сумарні дані доступні для використання через спеціалізовані Сховища, кошти data mining і OLAP; що ж стосується залежних вітрин даних, то тільки вони організовані за допомогою просторових моделей, як і у Ральфа Кимболла.

Таким чином, по суті справи архітектури відрізняються тільки способами поводження з атомарними даними: їх просторовою організацією у Кимболла і нормалізованої – у Інмона.

Друга принципова відмінність цих двох підходів, почасти випливає з першого, – питання фізичної організації Сховища. Якщо у Інмона Сховище даних – це фізично цілісний реально існуючий об'єкт, то Сховище Кимболла – швидше "віртуальний" об'єкт. Це колекція вітрин даних, які можуть бути просторово роз'єднаними.

Цими двома основними відмінностями в цілому і вичерпується принципова різниця між тією і іншою моделлю.

Закономірно виникає питання: так чия ж модель краще? Очевидно, що він не має однозначної відповіді. У цілому обидва ці підходи сходяться в головному – у необхідності сучасних засобів управління інформаційними потоками для прийняття своєчасних і обгрунтованих рішень при веденні

бізнесу і, відповідно, в необхідності створення відповідних структур для зберігання даних, їх координації та інтеграції. Вибір того чи іншого технічного рішення визначається потребами бізнесу і його конкретними особливостями.

Переваги і недоліки кожного з підходів безпосередньо випливають з їх архітектурних рішень. Вважається, що просторова організація з архітектурою "зірка" полегшує доступ до даних і вимагає менше часу на виконання запитів, а також спрощує роботу з атомарними даними. З іншого боку, прихильники підходу Білла Інмона критикують цю схему за відсутність необхідної гнучкості і вразливість структури, вважаючи, що в просторово організовані атомарні дані важче вносити необхідні зміни.

Реляційна схема організації атомарних даних уповільнює доступ до даних і вимагає більше часу для виконання запитів в силу різної організації атомарних і сумарних даних. Але, з іншого боку, ця схема надає широкі можливості для маніпулювання атомарними даними та зміни їх формату та способу подання в міру необхідності.

Підводячи підсумок, можна сказати, що, незважаючи на здаються глибокі відмінності між двома підходами до архітектури сховищ даних, це відмінності в основному технічного плану, а в цілому Сховища обох типів виявляються досить схожими за своїми функціями та завданням, які можна вирішувати з їх допомогою.

1.3.3 Гібридний підхід

Деякі організації використовують так званий гібридний підхід, намагаючись поєднати те краще, що є в обох методах. Як видно з рис. 1, гібридне Сховище даних поєднує розглянуті в попередній статті моделі (див. рис. 1.1 і 1.2 в [3]). Воно включає нормалізоване Сховище CIF і просторове Сховище атомарних і сумарних даних на основі архітектури шини Кимболла.

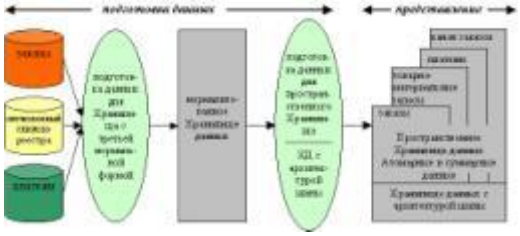

Рис.1.3 - Гібрид нормалізованого та просторового Сховищ даних

Варто підкреслити, що якщо остаточне уявлення даних прийнятно для використання, то такий підхід можна вважати життєздатним. Але подвійна робота з підготовки та зберігання атомарних даних супроводжується суттєвими додатковими витратами і затримками. Тому, ймовірно, варто витратити інвестиції в ресурси і технології на те, щоб відповідним чином представити додаткові ключові показники продуктивності для бізнесу.

Безумовно, якщо в організації спочатку було створено нормалізоване Сховище даних, а потім виникла необхідність у розвитку можливостей представлення даних, щоб продемонструвати їх цінність, то гібридний підхід допоможе вигідно використовувати вже зроблені інвестиції.

1.4 Побудова Вимірів (Dimensions)

Відстеження змін значень аналітичних вимірювань в сховище даних вирішується шляхом застосування механізму повільно мінливих вимірювань (Slowly Changing Dimensions, SCD).

У даній роботі розглядаються найбільш популярні типи повільно мінливих вимірювань - SCD Type 1, SCD Type 2 і SCD Type 3.

Повільно мінливі вимірювання в базі даних реалізуються у вигляді звичайних таблиць, в які додається ряд службових стовпців, що дозволяє реалізувати логіку відстеження змін даних.

Для всіх типів повільно змінювальних вимірювань принцип роботи з записами ґрунтується на попередньому визначенні дії, виробленого із записом (вставка, зміна, видалення), і подальшої логіці обробки записи в залежності від виробленого з нею дії.

Залежно від дії, виробленого із записом (вставка, зміна, видалення),

для запису може бути визначений один з наступних статусів:

• На вставку - запис нова і її необхідно додати в таблицю.

• На зміну - запис існує в таблиці, але в якихось полях змінилися вміст.

• На видалення - запис існує в таблиці, але тепер її необхідно видалити з неї.

Зазвичай реалізація повільно мінливих вимірювань проводиться в підсистемі ETL корпоративного сховища даних, а визначення статусу записи проводиться в момент захоплення змін даних.

1.4.1 Повільно змінювані виміри типу 1 (slowly change dimensions type 1)

До повільно змінюваних вимірювань першого типу відносяться ті вимірювання, в яких не підтримується відстеження змін даних в часі, тобто значення полів записів в разі їх зміни просто оновлюються.

Дії, вироблені з записами таблиці SCD Type 1 в залежності від їх статусу, представлені в таблиці нижче.

Табл.1.1 — Статус запису SCD type 1

Статус запису	Дія
На додавання	Записи присвоюється наступний по порядку унікальний ідентифікатор. Запис додається в таблицю.
На зміну	Запис змінюється.
На видалення	Ніяких дій над записом не проводиться. Видаляти запис з таблиці не можна, тому що до неї можуть бути «прив'язані» фактичні дані.

Наведемо наочний приклад відстеження змін для випадку, коли змінюється вміст одного з полів записи (вставка і видалення не представляють інтересу).

Припустимо, існує таблиця, в якій зберігається інформація по клієнтських даних (CST), що складається з 2-х полів: ID - первинний ключ

запису і NAME - найменування клієнта.

Табл.1.2 - Клієнтські дані

D	NAME
	ІП Іванов

У разі зміни існуючого найменування «ІП Іванов» на «ТОВ" Іванов і Ко. "» Запис в таблиці зміниться наступним чином.

Табл.1.3 - Нові клієнтські дані

D	NAME
	ТОВ "Іванов і Ко."

1.4.2 Повільно змінювані виміри типу 1 (slowly change dimensions type 2)

До повільно змінюваних вимірювань другого типу відносяться ті вимірювання, в яких підтримується відстеження змін даних в часі таким чином: стара запис позначається як втратила актуальність, і додається новий запис з тим же ідентифікатором, але вже з оновленими полями.

Структура таблиці типу SCD Type 2 крім основних її полів, що несуть інформацію для користувача, включає в себе наступні поля:

ID - унікальний ідентифікатор запису (входить до складу первинного ключа таблиці);

EFCT_DT - дата, з якої запис дійсна (входить до складу первинного ключа таблиці);

END_DT - дата, до якої запис дійсна (для всіх активних записів вона встановлена за замовчуванням, наприклад, в 01.01.2999);

IS_ACT_IND - індикатор активної записи: 1 - активна; 0 - не активна;

IS_DEL_IND - індикатор віддаленої запису: 1 - видалена; 0 - не

видалена.

Дії, вироблені з записами таблиці SCD Type 2 в залежності від їх статусу, представлені в таблиці нижче.

Табл.1.4 — Статус Запису SCD type 2

Статус запису	Дія
На додавання	Полю ID присвоюється наступний по порядку унікальний ідентифікатор. Полю EFCT_DT присвоюється поточна дата (SYSDATE). Полю END_DT присвоюється дата 01.01.2999. Полю IS_ACT_IND присвоюється 1. Полю IS_DEL_IND присвоюється 0. Запис додається в таблицю.
На зміну	Полю END_DT змінилася записи присвоюється поточна дата (SYSDATE). Полю IS_ACT_IND змінилася записи присвоюється 0. Додається новий запис в таблицю, у якій: Полю ID присвоюється такий же ідентифікатор, як і у зміненій записи. Полю EFCT_DT присвоюється поточна дата (SYSDATE). Полю END_DT присвоюється дата 01.01.2999. Полю IS_ACT_IND присвоюється 1. Полю IS_DEL_IND присвоюється 0.
На видалення	Полю END_DT присвоюється поточна дата (SYSDATE). Полю IS_ACT_IND присвоюється 0. Полю IS_DEL_IND присвоюється 1.

Наведемо наочний приклад для випадку, коли запис змінюється.

Табл.1.5 - Зміни клієнтських даних

ID	NAME	EFCT_DT	END_DT	IS_ACT_IND	IS_DEL_IND
1	ІП Іванов	01.10.2010	01.01.2999	1	0

У разі зміни існуючого найменування «ІП Іванов» на «ТОВ" Іванов і Ко. "» Записи в таблиці будуть виглядати наступним чином.

Табл.1.6 - Загальні клієнтські дані

ID	NAME	EFCT_DT	END_DT	IS_ACT_IND	IS_DEL_IND
1	ІП Іванов	01.10.2010	10.11.2010	0	0
1	ТОВ "Іванов и Ко."	10.11.2010	01.01.2999	1	0

1.4.3 Повільно змінювані виміри типу 1 (slowly change dimensions type 3)

До повільно змінюваних вимірювань третього типу відносяться ті вимірювання, в яких підтримується відстеження змін даних в часі шляхом додавання в структуру таблиць полів, що зберігають попередні значення.

Якщо SCD Type 2 дозволяє відстежувати необмежену кількість змін, то в SCD Type 3 кількість відслідковуються змін обмежується кількістю додаткових полів.

Структура таблиці типу SCD Type 3 крім основних її полів, що несуть інформацію для користувача, включає в себе наступні поля:

NAME_OLD - попереднє значення поля NAME, значення до зміни (NAME наведено як приклад, відстежувати зміни можна будь-яких полів таблиці);

NAME_UPD_DT - дата зміни значення поля NAME.

Дії, вироблені з записами таблиці SCD Type 3 в залежності від їх статусу, представлені в таблиці нижче.

Табл.1.7 — Статус Запису SCD type 3

Статус запису	Дія
На додавання	Записи присвоюється наступний по порядку унікальний ідентифікатор (в поле ID). В поле NAME присвоюється завантажувати значення. В поле NAME_OLD присвоюється значення за замовчуванням, наприклад «NA». Полю NAME_UPD_DT присвоюється поточна дата (SYSDATE) або дата за замовчуванням, наприклад 01.01.1900. Запис додається в таблицю.
На зміну	В поле NAME_OLD присвоюється значення з поля NAME. В поле NAME присвоюється завантажувати значення.
На видалення	Ніяких дій над записом не проводиться.

Приведемо зразок для випадку, коли запис змінюється.

Табл.1.8 - Первинні клієнтські дані

ID	NAME	NAME_OLD	NAME_UPD_DT
1	ІП Іванов	NA	01.10.2010

У разі зміни існуючого найменування «ІП Іванов» на «ТОВ" Іванов і Ко. "» Записи в таблиці будуть виглядати наступним чином.

Табл.1.9 - Оновлені клієнтські дані

ID	NAME	NAME_OLD	NAME_UPD_DT
1	ТОВ "Іванов и Ко."	ІП Іванов	10.11.2010

Висновки

Умови розвитку сучасного інформаційного простору вказують на необхідність опрацювання великих обсягів даних структурованого, слабко-структурованого і неструктурованого характеру, які знаходяться у

різнорідних джерелах: реляційних і багатовимірних базах даних, базах даних XML і NoSQL, структурованих і неструктурованих текстових файлах, медіафайлах тощо. Проблема ефективної роботи з даними є багатоаспектною і завжди привертала увагу вчених і фахівців-практиків та знайшла своє відображення в працях учених, зокрема N. Debbarma, G. Nath, H. Das [1], J.F. Kimpel [2], J.O.Chan [3], M. Jarke,M. Jeusfeld, C. Quix, P. Vassiliadis [4], Y. Yuan, R.L.X.Zhang [5], M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis [6], N.Rahman [7], L. Song, Y. Zhan, Y. Li [8], A. Gosain, S. Sabharwal, R. Gupta [9], C.Blanco, I.de Guzmán, E. Fernández-Medina [10].

У даному розділі представлений аналіз робіт, в яких описано підходи, моделі, методи та засоби побудови сховищ даних, сформульовано мету і завдання дослідження, а також концепцію побудови корпоративних сховищ.

РОЗДІЛ 2

ОПИС ПОБУДОВИ ЕТЛ ПРОЦЕСУ ЗА ДОПОМОГОЮ ORACLE DATA INTEGRATOR

2.1 Поняття та архітектура Oracle Data Integrator

Oracle Data Integrator (ODI) - це інтеграційна платформа корпоративного рівня, яка забезпечує вилучення, перетворення і завантаження даних з різноманітних джерел: баз даних, файлів і інших джерел (наприклад, LDAP каталогів або WEB-сервісів).

Oracle Data Integrator заснований на архітектурі ELT (Extract - Load - Transform), відповідно до якої вся навантаження по перетворенню даних покладається на СУБД. При цьому досягається високий рівень продуктивності за рахунок використання максимуму можливостей і особливостей всіх використовуваних в процесі обробки даних СУБД і мінімізації передачі даних по локальній мережі. Так само цей підхід забезпечує більш раціональне використання вкладень в устаткування і ліцензії за рахунок використання одних і тих самих потужностей для вирішення як завдань користувачів так і завантаження (з рознесенням за часом). Схематично типовий (рекомендований) процес завантаження даних виглядає наступним чином:

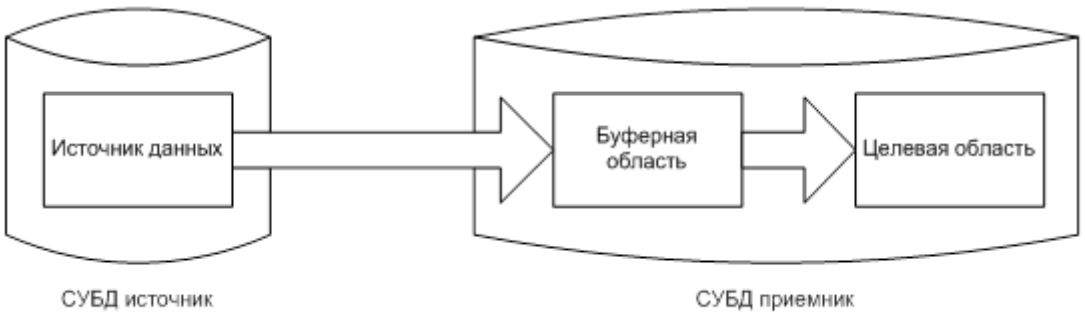

Рис.2.1- Принцип роботи ODI

Продукт ODI, будучи родоначальником ELT-підходу, сам не виробляє перетворень даних. Замість цього він зберігає тільки метаописания

перетворень, за якими генерує оптимізований код в діалекті мови SQL (або його розширень - PL / SQL, T-SQL і т. Д.) Конкретного джерела / одержувача даних. При цьому використовуються всі можливості і особливості цієї мови для побудови швидкого та ефективного коду, створюються умови для того, щоб ELT-процеси виконувалися в пакетному режимі, проводиться розпаралелювання обробки даних і забезпечується автоматична перевірка якості даних.

При цьому ODI не використовує проміжного шару для перетворення даних, т. Е. Область обробки даних (staging area) може знаходитися в базі даних як джерела, так і одержувача.

Продукт ODI працює в різнорідних середовищах, підтримує великий ряд джерел, включаючи різні реляційні бази даних, бізнес-додатки, плоскі файли, XML-файли, JMS-черзі, багатовимірні бази даних і багато іншого. В цілому цей продукт складається з декількох компонентів, які працюють з єдиним централізованим репозиторієм метаданих (metadata repository). Ці компоненти - графічні модулі (graphical modules), компоненти часу виконання (runtime components) і Web-інтерфейс - разом з іншими просунутими функціями роблять його «легкої», досконалої інтеграційної платформою.

Архітектура ODI організована навколо модульного сховища, який доступний компонентам, графічним модулям і агентам виконання (execution agents), цілком написаним на Java, в режимі клієнт-сервер. ODI є «чистим» Java додатком, що працює на будь-якій платформі, яка підтримує JVM 1.5 (J2SE), включаючи такі популярні як Windows, Linux, HP-UX, Solaris, AIX, Mac OS і багато інших. Крім того, архітектура ODI включає Web-додаток - Metadata Navigator, яке дозволяє отримувати доступ до інформації, в тому числі до сховища, через Web-інтерфейс. Він дозволяє проводити аудит, контроль та адміністрування всього процесу завантажень даних, а також runtime-управління.

Тепер подивимося на те, які графічні модулі входять до складу нового рішення і які функції вони виконують. Їх всього чотири: дизайнер (designer), оператор (operator), топологія (topology manager) і безпеку (security

manager).

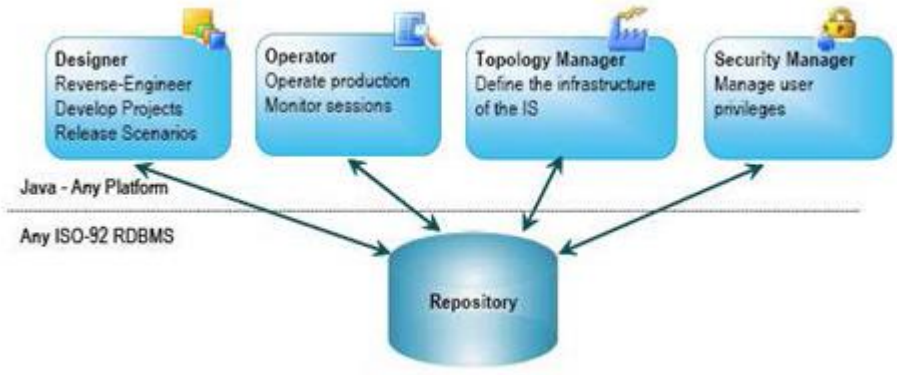

Рис.2.2 - Графічні модулі і репозиторій

Функції графічних модулів:

- Designer визначає декларативні правила (declarative rules) для перетворення даних і забезпечення їх цілісності (data integrity). Саме тут проявляється оригінальність підходу в меппінга даних, коли відбувається поділ правил трансформації (transformation rules) і інтеграції даних (data integrity), тобто чітко поділяються бізнес-правило «що робити» і імплементація цього правила - «як робити». Вся розробка проекту відбувається в цьому модулі, саме тут визначаються і імпортуються метадані баз даних і додатків. Дизайнер використовує метадані і правила генерації сценаріїв для виробничого середовища. Цей модуль є ключовим для розробників і адміністраторів метаданих. Слід зазначити, що єдиний репозиторій метаданих і оригінальний інструментарій меппінга даних дозволяють багаторазово використовувати процеси перетворення і перевірки даних. Можна окремо зберігати бізнес-правила, які можуть бути одним і тими ж для різних баз даних і їх конкретні імплементації, що враховують особливості джерела інформації. Виходить, що конкретні особливості бази даних проявляються на більш пізніх етапах розробки і в якійсь мірі вони відв'язані від бізнес-логіки

- процесу інтеграції. В цілому такий підхід проявляється у багатьох системах, коли створюються шаблони перетворень і складові оператори, які потім застосовуються послідовно один над одним, поступово ускладнюючи логіку роботи.

- Operator управляє і спостерігає за виробничим середовищем. Він розроблений для операторів цього середовища і показує журнали виконання (execution logs) з підрахунком помилок, числом опрацьованих рядків, статистикою виконання, кодом, який виконується в даний момент, і так далі. На етапі проектування (design time) розробники можуть використовувати модуль Operator для цілей налагодження;
- Topology Manager визначає фізичну і логічну архітектуру інфраструктури. Сервери, схеми і агенти реєструються в головному (master) репозиторії через цей модуль, як правило, адміністраторами інфраструктури або проекту;
- Security Manager управляє профілями користувачів і привілеями їх доступу. Security Manager також призначає привілеї доступу до об'єктів і функцій (features). Цей модуль зазвичай використовується адміністраторами безпеки.

Всі модулі зберігають свою інформацію в централізованому сховищі.

До компонентів часу виконання відноситься планувальник (Scheduler Agent), який координує виконання сценаріїв. Виконання може бути запущено одним з графічних модулів, або вбудованим оброблювачем розкладів (build-in sheduler), або зовнішнім обробником розкладів (third-party scheduler). В рамках архітектури E-LT планувальник рідко виконує будь-які перетворення.Він вибирає код зі сховищ виконання (execution repository) і потім запитує сервери баз даних, операційні системи. Коли виконання завершено, планувальник змінює журнали виконання (execution logs) в репозиторії і потім формує звіти з повідомленнями про помилки і статистикою виконання. Користувачі можуть переглядати журнали

виконання з модуля, Operator або Web-інтерфейсу Metadata Navigator. Важливо відзначити, що хоча планувальник може діяти як «двигун» перетворень, він рідко використовується з цією метою.

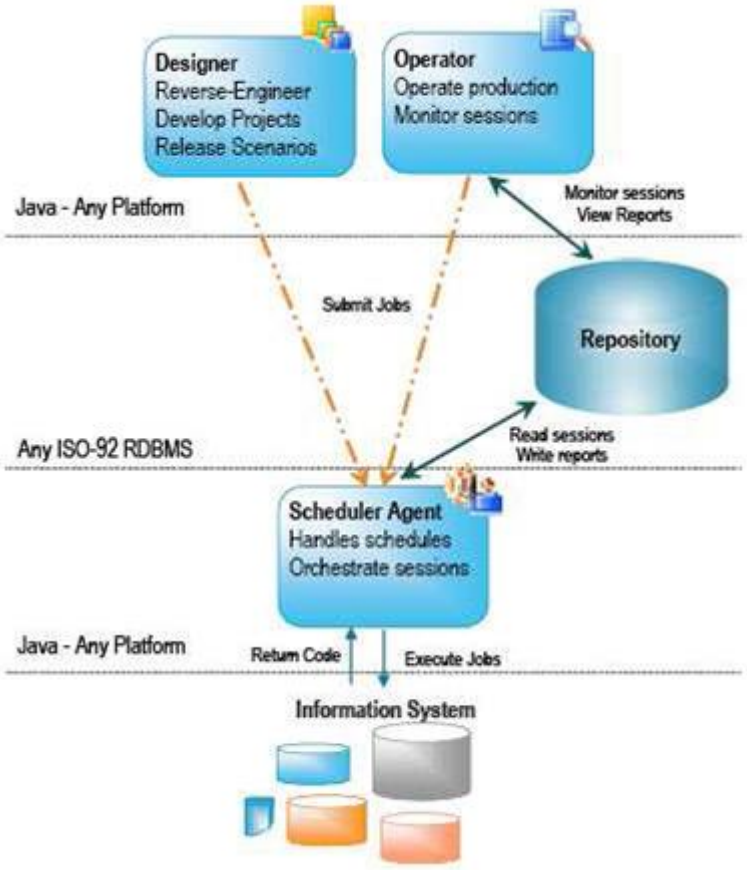

Рис.2.3 - Компоненти часу виконання.

2.2 Репозитарії Oracle Data Integrator

Сховище складається з головного (або майстер, master) сховища та кількох робітників (work) репозиторіїв. Ці репозиторії є базами даних, які керуються засобами реляційних СУБД. Всі об'єкти, які с застосуванням модулів конфигурируются, розробляються або використовуються, зберігаються в одному з цих репозиторіїв і доступні в режимі клієнт-сервер для різних компонентів архітектури.

Зазвичай є один головний репозиторій, який містить інформацію про безпеку (призначені для користувача профілі і привілеї), топологічну інформацію (визначення технологій і серверів) і версії об'єктів. Для ведення інформації, що зберігається в головному репозиторії, використовуються Topology Manager і Security Manager. Всі модулі мають доступ до головного сховища, так як всі вони зберігають інформацію про топології і безпеки в ньому.

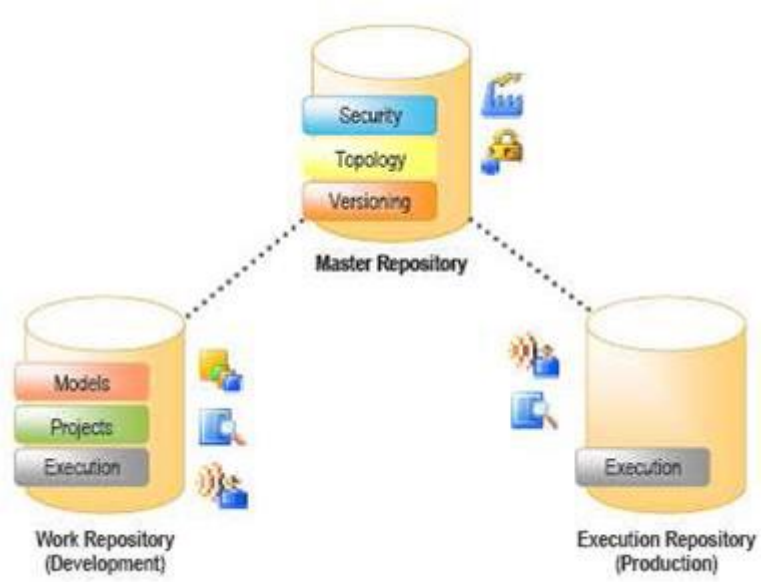

Рис.2.4 - Головний репозиторій і робочі репозиторії.

Об'єкти проектів зберігаються в робочих репозиторіях. Кілька робочих репозиторіїв можуть співіснувати на одній і тій же установці. Це корисно для ведення окремих середовищ або відображення особливих версій життєвого циклу - наприклад, середовища розробки (development), кваліфікування (qualification) і виробниче середовище.

Робочий репозиторій зберігає інформацію по таким об'єктам:

Моделі (Models) - включаючи області зберігання даних (datastores), колонки (columns), обмеження цілісності даних (data integrity constraints), перехресні посилання (cross references) та походження даних (data lineage);

Проекти (Projects) - включаючи декларативні правила, пакети (packages), процедури, папки, модулі знань (knowledge modules) і змінні

(variables);

Інформація часу виконання (Runtime information) - включаючи сценарії, інформацію розкладів і журнали.

Користувачі працюють з контентом робочого сховища, використовуючи модулі Designer і Operator. Робочі репозиторії також доступні під час виконання агентам.

Коли робочий репозиторій використовується тільки для зберігання інформації, необхідної для виконання (як правило, це має місце для виробничих середовищ), він називається репозиторієм виконання (execution repository). Цей репозиторій жлступен під час виконання агентам і через інтерфейс модуля Operator. Важливо пам'ятати, що всі робочі репозиторії завжди під'єднані до одного і тільки одного головного сховища.

Metadata Navigator (Навігатор метаданих) - це додаток для середовища Java 2 Enterprise Edition (J2EE), яке забезпечує доступ через Web до репозиторіїв. Воно дозволяє користувачам переглядати об'єкти, включаючи проекти, моделі та журнали виконання. Metadata Navigator може бути встановлений на сервер додатків, такий як Oracle Container for Java (OC4J) або Apache Tomcat. Бізнес-користувачі, розробники, оператори і адміністратори можуть використовувати Metadata Navigator через Web-браузер. Через Web-інтерфейс цієї програми користувачі можуть побачити карти потоків (flow maps), знайти джерела всіх даних і навіть "просвердлити" (drill down) до рівня показника (field level), щоб зрозуміти перетворення, які використовуються для побудови цих даних. Вони можуть також запускати сценарії і стежити за ними з Web-браузера через Metadata Navigator.

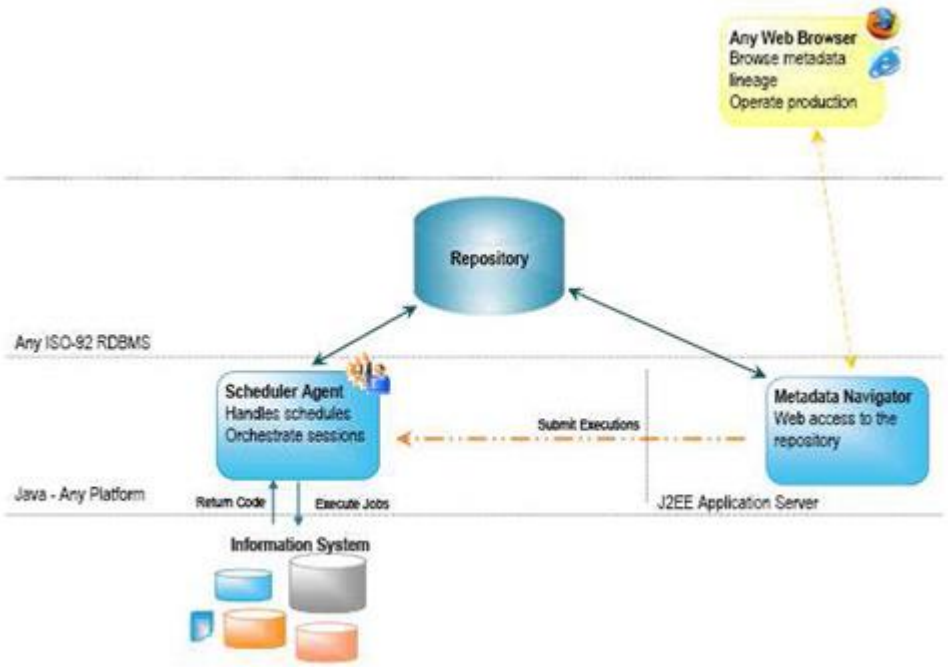

Рис.2.5 - Схема функціонування Metadata Navigator

Використовуючи Metadata Navigator, користувачі можуть отримувати доступ до метаданих і виконувати їх з Web-браузера.

2.3 Модулі Oracle Data Integrator

В архітектуру ODI закладений механізм розширення можливостей продукту шляхом підключення нових модулів знань (Knowledge Modules, KMs), які дозволяють зберігати специфічні (шаблонні) для даної платформи конструкції. Фактично, модулі знань є «плагінами», реалізують кращі практики завантаження і обробки даних для певного джерела даних або цільової СУБД.

Пошагове розділення функцій здійснюється за допомогою так званих модулів знань (Knowledge Module). Модулі знань представляють собою шаблони генерації коду на основі розроблених правил завантаження. У ODI виділяються наступні типи модулів знань:

Loading Knowledge Modules - модулі знань, призначені для забезпечення завантаження даних в ситуації коли джерело і приймач даних не є однією БД (в тому числі коли вони відносяться до різних СУБД). Наприклад, "LKM File to MSSQL (BULK)" - завантаження файлу в БД MS

SQL Server за допомогою bulk insert або "LKM MSSQL to Oracle (BCP / SQLLDR)" - вилучення даних з БД MS SQL Server за допомогою BCP і завантаження в БД Oracle за допомогою SQL * Loader.

Check Knowledge Module - модулі знань, призначені для виконання перевірок даних і обробки даних, які не пройшли перевірку. Перевірки можуть бути як виконуваними в довільний час по всій таблиці, так і виконуваними в процесі завантаження порції даних. Результатом обробки може бути як позначка помилкових записів і продовження процесу, так і зупинка процесу в разі критичності помилки.

Integration Knowledge Module - модулі знань, призначені для виконання завантаження даних безпосередньо в цільові таблиці. Наприклад, "IKM Oracle Incremental Update (MERGE)" - обчислення змінилася порції даних і завантаження її в цільову таблицю за допомогою оператора merge або "IKM Oracle Slowly Changing Dimension" - підтримка повільно мінливих вимірювань з автоматичним формуванням історичних версій даних.

Journalization Knowledge Module - модулі знань, призначені для відстеження даних, що змінилися. Наприклад, "JKM MSSQL Consistent" консистентні журнал даних в БД MS SQL Server за допомогою тригерів або "JKM MSSQL to Oracle Consistent (OGG)" - захоплення змін з БД MS SQL Server і перенесення в БД Oracle за допомогою Oracle Golden Gate.

Service Knowledge Module - модулі знань, призначені для генерації java класів, що реалізують web-сервіси доступу до даних.

Reverse-engineering Knowledge Module - модулі знань, призначені для читання метаданих (структури) джерел / приймачів даних. Наприклад, "RKM File (FROM EXCEL)" читання структури файлу Excel.

У стандартну поставку входить більше 150 модулів знань. Всі модулі знань є відкритими, в них можна вносити власні зміни для досягнення бажаного результату. Так само можливо розробляти власні модулі знань в разі, коли комбінування і зміна стандартних виявляється недостатнім.

Застосування різних модулів знань дозволяє реалізувати різні сценарії завантаження одних і тих же даних без необхідності розробки процедур під кожен сценарій. Наприклад, при первинне завантаження можуть

використовуватися модулі, передають великий обсяг даних через файли, а при штатній інкрементальною завантаженні модулі, що працюють з прямим підключенням до БД. Логічна схема завантаження (маппінг полів) при цьому залишається незмінною.

Модулі знань є найголовнішою складовою ODI (особливо LKM та IKM) для розробника ETЛ процесів, оскільки саме від побудови цієї логіки буде залежати правильність завантаження, трансформації та отримання даних. У третьому розділі цієї роботи я продемонструю власну розробку модуля знань за типом SCD type 2, який реалізую дуже складну, але найважливішу логіку при побудові ODS рівня сховища даних.

РОЗДІЛ 3

РОЗРОБКА ПОБУДОВИ ЕТЛ ПРОЦЕСУ НА ПРИКЛАДІ ПОБУДОВИ ODS РІВНЯ

3.1 Основне призначення ODS рівня

ODS являє собою відокремлену середу, де зазвичай розробники ЕТЛ процесів зберігать або оперативну інформацію або оперативну та історичну інформацію. Взагалі цей рівень слугує для того, щоб складувати інформацію з різних джерел даних та зберігати у тому вигляді як вони є на різних джерелах, будь то різноманітні СУБД, або плоскі файли, тощо. На перший погляд, операційний склад даних дуже схожий на сховище за структурою і змістом. Звичайно, за деякими характеристиками ODS і сховище даних дуже схожі, але ODS має ряд властивостей, які істотно відрізняють його від сховища. Як ODS, так і сховище даних є предметно-орієнтованим інтегрованим набором даних. З цієї точки зору вони схожі, оскільки як в одному, так і в іншому випадку дані повинні бути завантажені з транзакційних систем. Але на цьому їх схожість закінчується. ODS містить дані, що змінюються, тоді як в сховищі дані після завантаження не змінюються. Інша відмінність полягає в тому, що операційний склад містять тільки дані, актуальні на поточний момент часу, тоді як в сховищі містяться як поточні, так і історичні дані. При цьому актуальність даних в сховищі значно нижче, ніж в операційному складі. Як правило, в сховищі містяться дані, завантажені протягом останніх 24 годин, тоді як актуальність даних в ODS може вимірюватися секундами. Ще однією відмінністю ODS від сховища є те, що в ньому містяться тільки детальні дані, тоді як сховище містить як детальні, так і агреговані дані.

Крім завдань оперативного аналізу даних ODS можуть використовуватися при інтеграції транзакційних систем. У разі використання

ODS інформаційні потоки між системами будуються таким чином, що обмін даними відбувається між транзакционной системою і ODS, а не безпосередньо між транзакційними системами. Загальна кількість інтерфейсів між системами, таким чином, скорочується. Як приклад подібного ODS можна привести довідник контрагентів. Як правило, інформація про контрагентів міститься в більшості транзакційних систем.

При цьому, в кожній інформаційній системі враховується якийсь свій блок інформації і виділити одну систему в якості головної (в якій будуть заводитися контрагенти, а потім поширюватися в інші системи) не завжди можливо. Для вирішення даного завдання створюється ODS, в якій з транзакційних систем потрапляє інформація про контрагентів (з кожної системи свій блок інформації). Таким чином, отримуємо єдиний довідник контрагентів, в якому міститься максимально повна інформація.

При цьому в ODS повинен бути реалізований механізм пошуку «двійників», за яким система буде визначати, що такий контрагент вже існує. Найпростішим прикладом такого пошуку може бути порівняння якогось унікального коду, наприклад ІНН.

3.2 Побудова ODS рівня за допомогою Oracle Data Integrator

Як вже було зазначено побудова ODS рівня засобами Oracle Data Integrator має свою специфіку, особливості, переваги та недоліки. Кожен шаг цього алгоритму формую один великий ланцюг, який дозволяю реалізувати цей складний алгоритм. Схема БД, яка вже побудована за допомогою Oracle Data Integrator

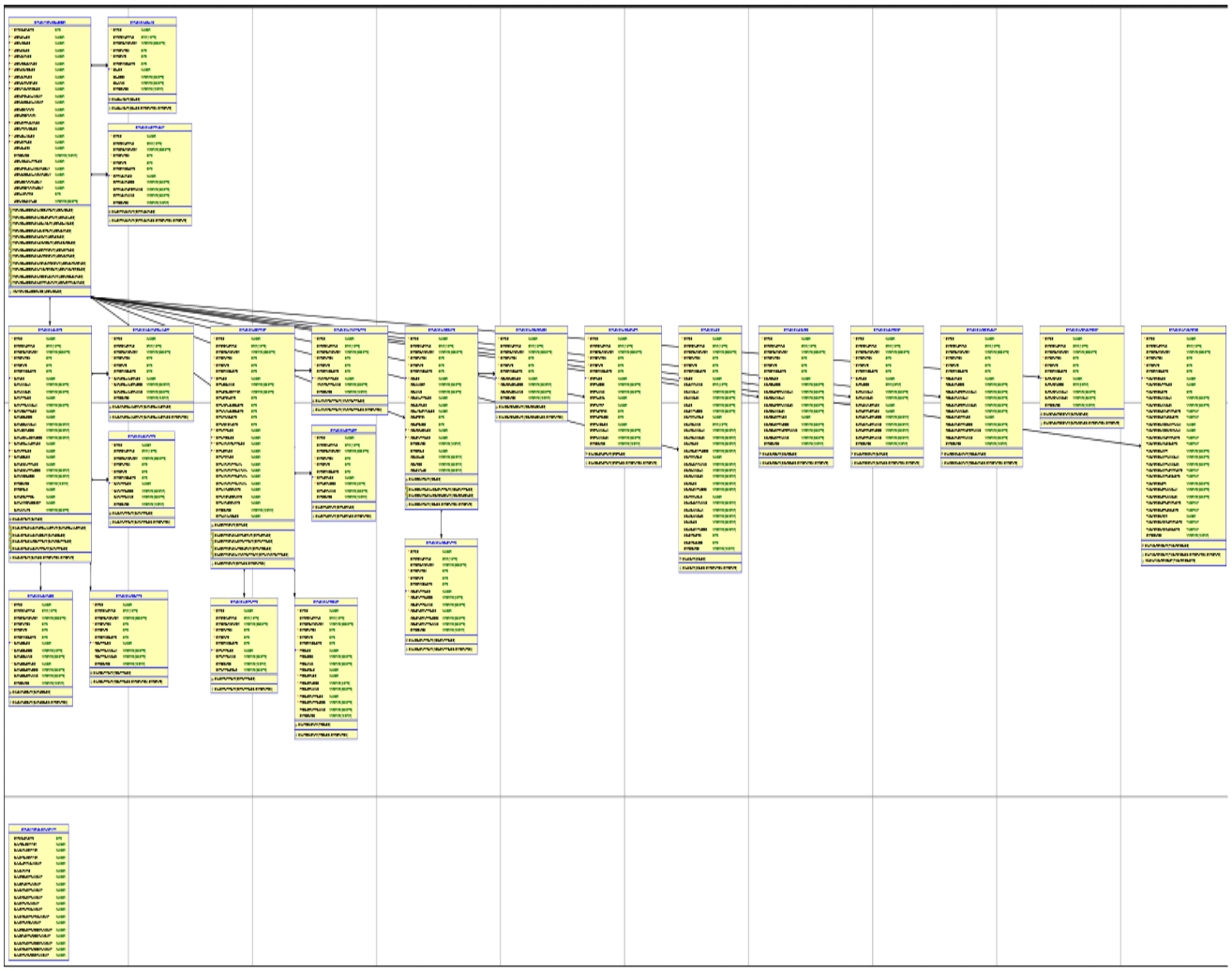

Рис.3.1 - Схема Сніжинка

Треба зазначити що, так як Oracle Data Integrator використовую власні зміні які потім Java замініє на ті зміні які використовуються в маппінгу програми. Давайте перерахуємо усі кроки які ми будемо розглядати у цьому алгоритмі:

1) Дроп тимчасової таблиці

drop table <%=odiRef.getTable("L", "COLL_NAME", "A")%>

При виконані цього кроку буде виконана така дія:

drop table C$_SRC_TABLE

2) Створюємо тимчасову таблицю

create table <%=odiRef.getTable("L", "COLL_NAME", "A")%>

(<%=odiRef.getColList("", "[COL_NAME]\t[DEST_WRI_DT] NULL", ",\n\t", "","")%>, DWH$SRC_HASH_ROW VARCHAR2(4000 char), DWH$CHANGE_TYPE CHAR(1), DWH$TIMESTAMP TIMESTAMP) NOLOGGING

Буде створена таблиця C$_SRC_TABLE, яка буде повторювати структуру таблиці джерела.

3) Вибираємо дані з сорс таблиці та завантажуемо у тимчасову таблицю

select * from

(SELECT <%=odiRef.getColList("", "[COL_NAME]", ",\n ", ",", "")%> ROW_NUMBER () OVER (PARTITION BY <%=odiRef.getColList("", "[COL_NAME]", ", ", "", "UK")%>

ORDER BY <%=odiRef.getColList("", "[COL_NAME]", ", ", "", "UK")%> DESC) RNUM from <%=odiRef.getFrom()%>

where (1=1)

<%=odiRef.getFilter()%>

<%=odiRef.getJoin()%>

<%=odiRef.getJrnFilter()%>

<%=odiRef.getGrpBy()%>

<%=odiRef.getHaving()%>

order by <%=odiRef.getColList("", "[COL_NAME]", ", ", "", "UK")%> DESC) WHERE RNUM=1

select <%=odiRef.getPop("DISTINCT_ROWS")%>

<%=odiRef.getColList("", "[EXPRESSION]\t[ALIAS_SEP] [COL_NAME]", ",\n\t", "", "")%> from <%=odiRef.getFrom()%>

```
where (1=1)

<%=odiRef.getFilter()%>

 <%=odiRef.getJoin()%>

 <%=odiRef.getJrnFilter()%>

 <%=odiRef.getGrpBy()%>

<%=odiRef.getHaving()%>

insert into <%=odiRef.getTable("L", "COLL_NAME", "A")%>

 (  <%=odiRef.getColList("",  "[COL_NAME]",  ",\n        ",  ",",  "")%>
DWH$SRC_HASH_ROW, DWH$TIMESTAMP  )

values  ( <%=odiRef.getColList("",  ":[COL_NAME]",  ",\n      ",  ",",  "")%>
DWH_STAGE.DWH_GET_HASH_F(<%=odiRef.getColList("","TO_CHAR(:[C
OL_NAME])", "\n   ||", "","(INS OR UPD) AND !UD3")%>)
```

Цей код використовується в для виборки та вставки в тимчасову таблицю, з використанням хєш-функції, яка відіграє важливу роль в цьому процесі. Завдяки хєш функції ми будемо вираховувати які сроки змінилися а які ні, що не роботи порівняння по усім колонками, тим паче це такаж звільняє нас від порівняння колонок, які мають null значення.

4) Завантаження сорс-таблиці у файл:

OdiSqlUnload "-FILE=<%=odiRef.getOption("DIR_PATH")%>/
<%=odiRef.getSrcTablesList("","[TABLE_NAME]","","")%>.dat"
"-DRIVER=<%=odiRef.getInfo("SRC_JAVA_DRIVER")%>"
"-URL=<%=odiRef.getInfo("SRC_JAVA_URL")%>"
"-USER=<%=odiRef.getInfo("SRC_USER_NAME")%>"
"-PASS=<%=odiRef.getInfo("SRC_ENCODED_PASS")%>"
"-FILE_FORMAT=VARIABLE" "-FIELD_SEP=,"
"-ROW_SEP=<% if (odiRef.getOption("UNIX_ENVIRONMENT").equals("1"))
{ %>n<%} else { %>rn<% } %>"
"-CHARSET_ENCODING=ISO8859_1" "-XML_CHARSET_ENCODING=ISO-8859-1"
"-FETCH_SIZE=<%=odiRef.getOption("FETCH_SIZE")%>"
"-QUERY=select <%=odiRef.getColList("", "[EXPRESSION]", ", ", "", "")%>
<%=odiRef.getColList("", "[EXPRESSION]", "||','||", "", "")%> from
<%=snpRef.getFrom()%>
where (1=1) <%=snpRef.getJoin()%>
<%=snpRef.getFilter()%> <%=snpRef.getGrpBy()%>
<%=snpRef.getHaving()%>"

```
OdiSqlUnload "-FILE=C:/ODI_SCEN_DMP/SNP_COL.dat"
"-DRIVER=oracle.jdbc.driver.OracleDriver"
"-URL=jdbc:oracle:thin:@localhost:1521:xe" "-USER=system"
"-PASS=<@=snpRef.getInfo("SRC_ENCODED_PASS") @>"
"-FILE_FORMAT=VARIABLE" "-FIELD_SEP=,""-ROW_SEP=\r\n"
"-CHARSET_ENCODING=ISO8859_1" "-XML_CHARSET_ENCODING=ISO-8859-1"
"-QUERY=select SNP_COL.I_COL, SNP_COL.I_TABLE, SNP_COL.COL_NAME,
SNP_COL.COL_HEADING, SNP_COL.COL_DESC, SNP_COL.SOURCE_DT,
SNP_COL.POS, SNP_COL.LONGC, SNP_COL.SCALEC, SNP_COL.FILE_POS,
SNP_COL.BYTES, SNP_COL.IND_WRITE, SNP_COL.COL_MANDATORY,
SNP_COL.CHECK_FLOW, SNP_COL.CHECK_STAT, SNP_COL.COL_FORMAT,
SNP_COL.COL_DEC_SEP, SNP_COL.REC_CODE_LIST,
SNP_COL.COL_NULL_IF_ERR, SNP_COL.DEF_VALUE, SNP_COL.INT_VERSION,
SNP_COL.IND_CHANGE, SNP_COL.FIRST_DATE, SNP_COL.FIRST_USER,
SNP_COL.LAST_DATE, SNP_COL.LAST_USER, SNP_COL.IND_IN,
SNP_COL.IND_OUT, SNP_COL.EXT_VERSION, SNP_COL.SCD_COL_TYPE,
SNP_COL.I_SRC_COL, SNP_COL.IND_WS_SELECT, SNP_COL.IND_WS_INSERT,
SNP_COL.I_TXT_COL_DESC, SNP_COL.IND_WS_UPDATE from
ODI_WORK.SNP_COL   SNP_COL where (1=1)   "
```

Рис.3.2 - Завантаження сорс-таблиці у файл

5) Генеруємо контрольний файл

```
SnpsOutFile "-File=<%=odiRef.getOption("DIR_PATH")%>/
<%=odiRef.getSrcTablesList("", "[TABLE_NAME].ctl", "", "")%>"
OPTIONS (
   SKIP=<%=odiRef.getSrcTablesList("", "[FILE_FIRST_ROW]", "", "")%>,
   ERRORS=<%=odiRef.getUserExit("LOA_ERRORS")%>,
   DIRECT=<%=odiRef.getUserExit("LOA_DIRECT")%>
   )
LOAD DATA
INFILE "<%=odiRef.getOption("DIR_PATH")%>/
<%=odiRef.getSrcTablesList("", "[RES_NAME]", "", "")%>" "str X'
<% if (odiRef.getOption("UNIX_ENVIRONMENT").equals("1")) { %>0A<%}
else {%>0D0A<% } %>'"
BADFILE "<%=odiRef.getOption("DIR_PATH")%>/
<%=odiRef.getSrcTablesList("", "[TABLE_NAME].bad", "", "")%>"
DISCARDFILE "<%=odiRef.getOption("DIR_PATH")%>/
<%=odiRef.getSrcTablesList("", "[TABLE_NAME].dsc", "", "")%>"
DISCARDMAX <%=odiRef.getUserExit("LOA_DISCARDMAX")%>
<% if (odiRef.getOption("CREATE_WORK_TABLES").equals("1")) { %>
INTO TABLE <%=odiRef.getTable("L", "COLL_NAME", "W")%>
FIELDS TERMINATED BY X'2C'
TRAILING NULLCOLS
(
   <%=odiRef.getColList("", "[CX_COL_NAME]
<? if (u0022[DEST_DT]u0022.equals(u0022DATEu0022))
{?> DATE "'+snpRef.getOption("DATE_FORMAT")+ "' <?}?>
<? if (u0022[DEST_DT]u0022.equals(u0022TIMESTAMPu0022))
{?> TIMESTAMP "'+snpRef.getOption("TIMESTAMP")+ "' <?}?>" , " ,nt",
"","")%>
)
<% } else {%>
INTO TABLE
```

<%=odiRef.getTargetTable("SCHEMA")%>.<%=odiRef.getTargetTable("TABLE_NAME")%>
FIELDS TERMINATED BY X'2C'
TRAILING NULLCOLS
(
<%=odiRef.getColList("", "[COL_NAME]
<? if (u0022[DEST_DT]u0022.equals(u0022DATEu0022))
{?> DATE "'+snpRef.getOption("DATE_FORMAT")+ "' <?}?>
<? if (u0022[DEST_DT]u0022.equals(u0022TIMESTAMPu0022))
{?> TIMESTAMP "'+snpRef.getOption("TIMESTAMP")+ "' <?}?>" , " ,nt", "","")%>

)
<% } %>

LKM | LKM OdiSqlUnload(File) to Oracle (SQLLDR)

Option	Value
DELETE_TEMPORARY_OBJECTS	<Default>:Yes
WORK_TABLE_OPTIONS	<Default>:NOLOGGING
CREATE_TARGET_TABLE	Yes
LOA_DIRECT	<Default>:FALSE
LOA_DISCARDMAX	<Default>:1
LOA_ERRORS	<Default>:0
COMPATIBLE	<Default>:9
DIR_PATH	C:/ODI_SCEN_DMP
CREATE_WORK_TABLES	<Default>:No
DELETE SQLLDR FILES	No
DATE_FORMAT	<Default>:YYYY-MM-DD HH24:MI:SS
TRUNCATE	<Default>:Yes
UNIX_ENVIRONMENT	No
TIMESTAMP_FORMAT	<Default>:YYYY-MM-DD HH24:MI:SS.FF

Рис.3.3 - Параметри SQL Loader

6) Викликаємо SQL Loader для завантаження

import os
if os.system(r"sqlldr userid=<%=odiRef.getInfo("DEST_USER_NAME")%>/<%=odiRef.getInfo("DEST_PASS")%>@<%=odiRef.getInfo("DEST_DSERV_NAME")%>

```
control=<%=odiRef.getOption("DIR_PATH")%>/
<%=odiRef.getSrcTablesList("[TABLE_NAME].ctl","")%>
log=<%=odiRef.getOption("DIR_PATH")%>/
<%=odiRef.getSrcTablesList("[TABLE_NAME].log", "")%> ") <> 0:
   openlog=open('<%=odiRef.getOption("DIR_PATH")%>/
   <%=odiRef.getSrcTablesList("[TABLE_NAME].log", "")%>','r')
   raise 'OS command has signalled errors.Please see the log for details ',
   openlog.readlines()
   openlog.close()
```

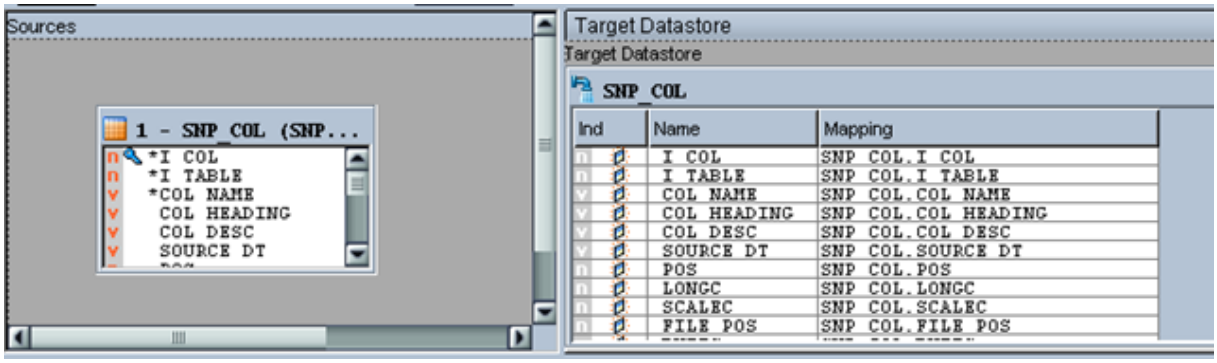

Рис.3.4 - Парсинг файлу

Як результат можемо перевірити код у Unix середовищі

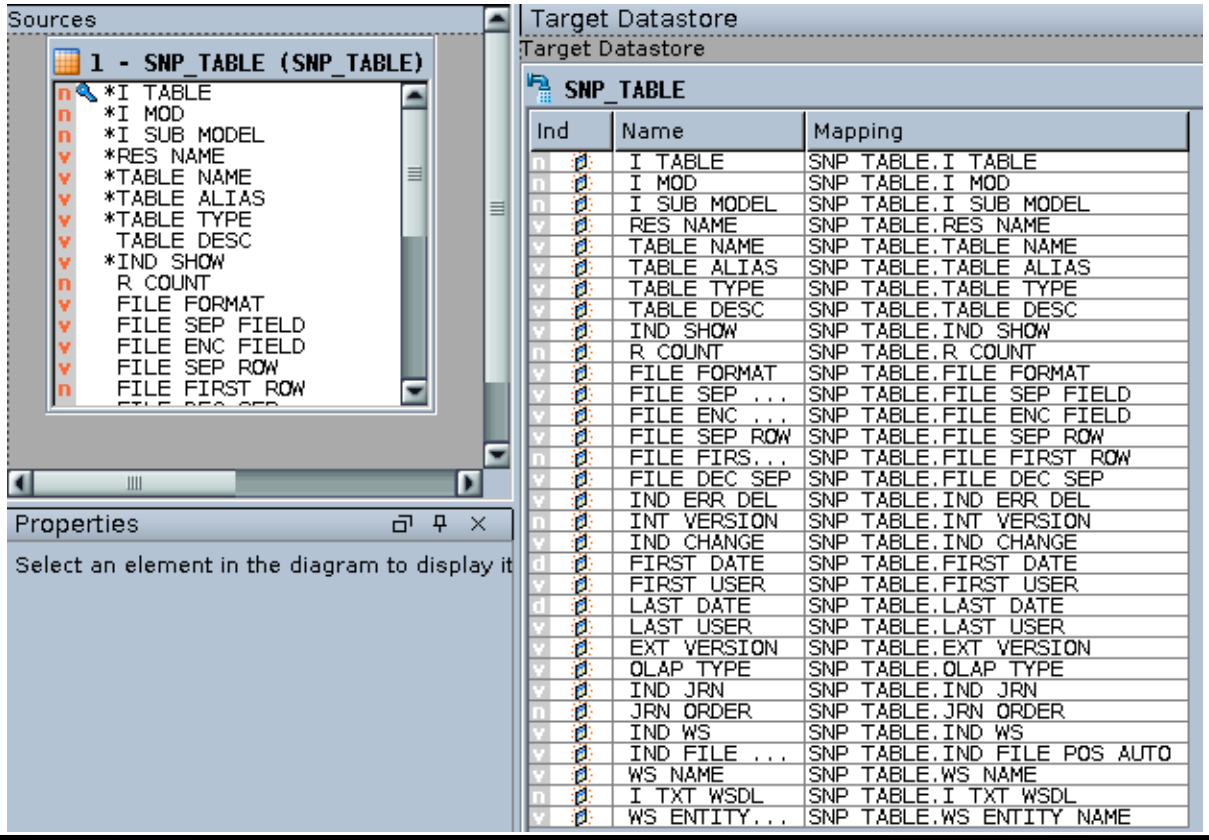

Рис.3.5- Парсинг ODI

Як виглядає створення маппінгу у Oracle Data Integrator:

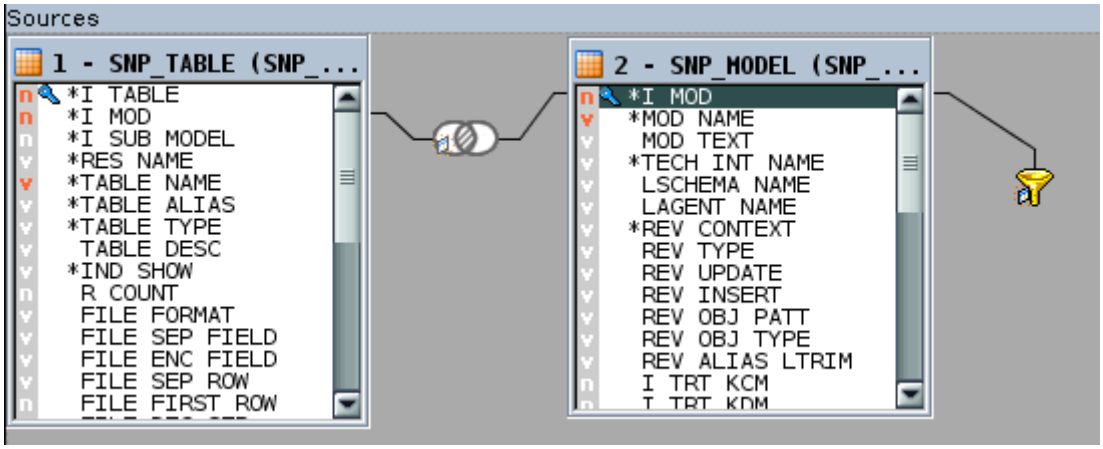

Рис.3.6 - Маппинг

ВИСНОВКИ

Метою роботи була реалізація сховища даних і процесу його наповнення, що виконується систематично. Для досягнення поставленої мети були виконані наступні дії:

1. Вивчення процесу планування, розробки та реалізації сховища даних. В ході вивчення була розглянута основна модель ХД, а саме модель «зірка», було отримано уявлення про сховище даних, його призначення і мети його створення.

2.Вивчення ЕТЛ засобу Oracle Data Integrator. Були отримані практичні навички по роботі з цією тулзою: створення диспетчерів сполук і їх параметризація, забезпечення потрібної послідовності виконання завдань за допомогою контейнерів, створення параметрів і їх використання в потоці даних при отриманні даних з джерела, застосування компонентів потоку даних для реалізації оновлення та видалення даних зі сховища.

3. Вивчення язика PL/SQL. В ході вивчення були освоєні основні команди мови і отримані наступні навички: створення таблиць, процедур і уявлень, використання змінних в запитах, маніпулювання даними, фільтрація даних.

4. Вивчення завантаження даних із неструктурованого файлу шляхом завантаженням методом SQL Loader. На підставі перенесених даних були створені зведена таблиця і зведена діаграма. В результаті були отримані: сховище даних, що дозволяє зберігати всі необхідні дані; ODI маппинг і скрипт PL/SQL, які реалізують ETL процес і в яких були враховані і реалізовані процеси

оновлення та видалення даних зі сховища; збережені процедури, що забезпечують аналіз даних зі сховища.

Було проведено порівняння методів наповнення сховища. За результатами порівняння був зроблений висновок, що для конкретного випадку, тобто для переміщення даних з бази даних OLTP в сховище найкраще використовувати PL/SQL скрипт, так як його продуктивність

набагато вище продуктивності ODI маппингу.

Надалі можна працювати над оптимізацією ODI маппингу, використовувати при розробці гібридний метод, тобто замінити завдання потоку даних на елемент, що виконує SQL скрипти. Виробляти витяг, перетворення і завантаження даних за допомогою збережених процедур.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Асєєв Г. Г. Концепція електронного сховища даних / Георгій Асєєв // Вісн. Кн. палати. — 2009. — № 2. — С. 28—30.
2. Вовчок В. А. Документальное хранилище на Web / В. А. Вовчок.
3. ГуляевА.И. Временные ряды в динамических базах данных. –М.: Радио и связь, 1989. –128с.
4. Деян Сарка, Матия Лах, Грега Йеркич. Microsoft SQL Server 2012. Реализация хранилищ данных //Русская редакция.2014.
5. Кетерін Дрюек (Katherine Drewek). "Сховища даних: завершення дебатів" (Data Warehousing: Our Great Debate Wraps Up).
6. Кетерін Дрюек (Katherine Drewek). "Сховища даних: підхід Білла Інмона" (Data Warehouse: Bill Inmon "s Vision).
7. Кетерін Дрюек (Katherine Drewek). "Сховища даних: підхід Ральфа Кимболла" (Data Warehouse: Ralph Kimball "s Vision).
8. Кетерін Дрюек (Katherine Drewek). "Сховища даних: реляційні та багатовимірні дані" (Data Warehousing: Relational vs. Multi-Dimensional Data).
9. Кетерін Дрюек (Katherine Drewek). "Сховища даних: схожість і відмінності підходів Білла Інмона і Ральфа Кимболла" (Data Warehousing: Similarities and Differences of Inmon and Kimball).
10. Марджі Росс (Margy Ross) і Ральф Кімболл (Ralph Kimball). "Різні думки" (Differences of Opinion).
11. Чернишова Т.В. Єдина методика створення інформаційного сховища // Вісник соціально-економічних досліджень. –2004. –No18 .–С. 404-408.
12. Шлеер С., Меллор С. Объектно-ориентированный анализ: моделирование мира в состояниях. -К.: Диалектика, 1993.3. Малахов Є.В. Вопросы организации иерархических информационных хранилищ// Перспективи.–1997. –No 1.–С.122 –123.
13. Hill W. Recommending and Evaluating Choices in a Virtual Community of Use / W. Hill [et al.] // Proc. Conf. Human Factors in Computing Systems

14. (CHI95), ACM Press. — New York, 2005. — P. 194—201.
15. Meta Data & knowledge Management: Meta Data Repository Myths, David Marco, DM Review, 2002
16. Pazzani M. A Framework for Collaborative, Content-Based and Demographic Filtering / M. Pazzani // Artificial Intelligence Review. — Dec., 2007. — P. 393—408.

ДОДАТКИ

Додаток А

Процедура завантаження даних

```
--наполнение таблицы
procedure ETL_S02_REF_DEAL_ACCOUNT_LINK(P_ON_DATE in date) is
 c_table_name CONSTANT VARCHAR2(30) := 'ETL$_S02_DACCL_GID';
 l_dt DATE := trunc(p_on_date);
 excPartitionNoExist exception;
 pragma exception_init(excPartitionNoExist, -02149);
BEGIN
 -- очистка таблицы
 EXEC_DDL('TRUNCATE TABLE ' || C_TABLE_NAME || ' REUSE STORAGE');

DWH_SECURITY.DWH_CONTEXT_EDIT_PKG.SET_CONTEXT('DWH_USER','DWH$START_DATE',TO_CHAR(P_ON_DATE,'DD.MM.YYYY'),'N');

 -- наполнение таблицы данными
insert /*+APPEND*/into ETL$_S02_DACCL_GID (DACCL_GID1, DACCL_GID2, DACCL_GID3, DACCL_GID4)
 select

/*************************************************************************
NAME: ETL_S02_REF_DACCL_GID_DS08_V
PERPOSE: «MAP_E000790_S02»
СВЯЗЬ СДЕЛКИ ОБСЛУЖИВАНИЯ КАРТОЧНОГО СЧЕТА С ЛИЦЕВЫМ СЧЕТОМ
REVISIONS:
VER DATE AUTHOR DESCRIPTION
------ --------- --------- -----------------------------------
1.0 04.02.2014 Setrina V.
1.1 24.02.2014 Setrina V.
```

```
        ************************************************************
**********/
        -- Индивидуальные счета учета в ОДБ
                AB.CARD_ACCT
               ,AB.CURR
               ,AB.CARD_KIND
               ,AB.DIC_ACCNT_BUH_TYPE_ID
              from S02_KONTS_ACCNT_BUH_CH AB
             where                                    AB.DWH$LOAD_DATE
=TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy')
              and AB.DWH$CHANGE_TYPE='I';

             commit;


        insert /*+APPEND*/into ETL$_S02_DACCL_GID (DACCL_GID1,
DACCL_GID2, DACCL_GID3, DACCL_GID4)
              --Шаблонные счета учета в ОДБ
        select
        AC.CARD_ACCT
        ,AC.CURR
        ,AC.CARD_KIND
        ,ATBA.DIC_ACCNT_BUH_TYPE_ID
        from S02_KONTS_ACCNT_CH AC
        inner join S02_KONTS_CLIENT CL
        on  CL.CLIENT    = AC.CLIENT
        and CL.CARD_KIND = AC.CARD_KIND
        and                          TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between CL.DWH$START_DATE and
CL.DWH$END_DATE
        and CL.DWH$ROW_STATUS = 'A'
        inner join S02_KONTS_CONDIT CND
```

```
        on  CND.CARD_KIND = AC.CARD_KIND
        and CND.COND_SET  = AC.COND_SET
        and CND.CURR      = AC.CURR
        and                          TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between CND.DWH$START_DATE and
CND.DWH$END_DATE
        and CND.DWH$ROW_STATUS = 'A'
              inner join S02_KONTS_ACCNT_TPL_BUH ATB
                on  ATB.BRANCH     = (case when CL.BRANCH between 64080
and 64101 then 64001 else CL.BRANCH end) --Отделение учета в ОДБ
                and ATB.CTYPE      = CL.CTYPE          -- 1-физ, 2-юр. лицо
                and ATB.RESIDENT   = CL.RESIDENT       -- R-резидент, N-
нерезидент
                and ATB.PRODUCT_ID = CND.PRODUCT_ID    -- Продукт
                and ATB.CURR       = AC.CURR           -- Валюта
                and  ATB.TERM_OVER   = NVL(AC.TERM_OVER, 1) -- 1 -
несрочный овердрафт, 2 - срочный
                and                  TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between ATB.DWH$START_DATE and
ATB.DWH$END_DATE
                AND ATB.dwh$row_status = 'A'
              inner join S02_KONTS_ACCNT_T_B_ACCNT ATBA
               on ATBA.ACCNT_TPL_BUH_ID = ATB.ACCNT_TPL_BUH_ID
        and                  TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between ATBA.DWH$START_DATE
and ATBA.DWH$END_DATE
        and ATBA.DWH$ROW_STATUS = 'A'
        where (AC.STATUS = 0 or AC.END_BAL<>0 or AC.DEBIT<>0 or
AC.CREDIT<>0)
        and                                    AC.DWH$LOAD_DATE
=TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy')
        and  AC.DWH$CHANGE_TYPE='I';
```

```
commit;

insert /*+APPEND*/into ETL$_S02_DACCL_GID (DACCL_GID1, DACCL_GID2, DACCL_GID3, DACCL_GID4)
select
AC.CARD_ACCT
,AC.CURR
,AC.CARD_KIND
,ATBA.DIC_ACCNT_BUH_TYPE_ID
from S02_KONTS_CLIENT_CH CL
inner join S02_KONTS_ACCNT AC
on  CL.CLIENT    = AC.CLIENT
and CL.CARD_KIND = AC.CARD_KIND
and (AC.STATUS = 0 or AC.END_BAL<>0 or AC.DEBIT<>0 or AC.CREDIT<>0)
and TO_DATE(SYS_CONTEXT('DWH_USER', 'DWH$START_DATE'),'dd.mm.yyyy') between AC.DWH$START_DATE and AC.DWH$END_DATE
and AC.DWH$ROW_STATUS = 'A'
inner join S02_KONTS_CONDIT CND
on  CND.CARD_KIND = AC.CARD_KIND
and CND.COND_SET  = AC.COND_SET
and CND.CURR      = AC.CURR
and TO_DATE(SYS_CONTEXT('DWH_USER', 'DWH$START_DATE'),'dd.mm.yyyy') between CND.DWH$START_DATE and CND.DWH$END_DATE
and CND.DWH$ROW_STATUS = 'A'

        inner join S02_KONTS_ACCNT_TPL_BUH ATB
         on  ATB.BRANCH     = (case when CL.BRANCH between 64080 and 64101 then 64001 else CL.BRANCH end) --Отделение учета в ОДБ
         and ATB.CTYPE      = CL.CTYPE          -- 1-физ, 2-юр. лицо
```

```
                and ATB.RESIDENT    = CL.RESIDENT          -- R-резидент, N-
нерезидент
                and ATB.PRODUCT_ID = CND.PRODUCT_ID      -- Продукт
                and ATB.CURR       = AC.CURR             -- Валюта
                and  ATB.TERM_OVER    = NVL(AC.TERM_OVER, 1) -- 1 -
несрочный овердрафт, 2 - срочный
                and                    TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between ATB.DWH$START_DATE and
ATB.DWH$END_DATE
                AND ATB.dwh$row_status = 'A'
              inner join S02_KONTS_ACCNT_T_B_ACCNT ATBA
               on ATBA.ACCNT_TPL_BUH_ID = ATB.ACCNT_TPL_BUH_ID
          and                          TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between ATBA.DWH$START_DATE
and ATBA.DWH$END_DATE
          and ATBA.DWH$ROW_STATUS = 'A'
          where                                    CL.DWH$LOAD_DATE
=TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy')
          and  CL.DWH$CHANGE_TYPE='I';

      commit;

      insert /*+APPEND*/into ETL$_S02_DACCL_GID (DACCL_GID1,
DACCL_GID2, DACCL_GID3, DACCL_GID4)
      select /*+ ORDERED */
      AC.CARD_ACCT
      ,AC.CURR
      ,AC.CARD_KIND
      ,ATBA.DIC_ACCNT_BUH_TYPE_ID
      from S02_KONTS_CONDIT_CH CND
      inner join S02_KONTS_ACCNT AC
      on  CND.CARD_KIND = AC.CARD_KIND
```

```
        and CND.COND_SET  = AC.COND_SET
        and CND.CURR      = AC.CURR
        and (AC.STATUS = 0 or AC.END_BAL<>0 or AC.DEBIT<>0 or AC.CREDIT<>0)
        and TO_DATE(SYS_CONTEXT('DWH_USER', 'DWH$START_DATE'),'dd.mm.yyyy') between AC.DWH$START_DATE and AC.DWH$END_DATE
        and AC.DWH$ROW_STATUS = 'A'
        inner join S02_KONTS_CLIENT CL
        on  CL.CLIENT    = AC.CLIENT
        and CL.CARD_KIND = AC.CARD_KIND
        and TO_DATE(SYS_CONTEXT('DWH_USER', 'DWH$START_DATE'),'dd.mm.yyyy') between CL.DWH$START_DATE and CL.DWH$END_DATE
        and CL.DWH$ROW_STATUS = 'A'
              inner join S02_KONTS_ACCNT_TPL_BUH ATB
                on  ATB.BRANCH     = (case when CL.BRANCH between 64080 and 64101 then 64001 else CL.BRANCH end) --Отделение учета в ОДБ
                and ATB.CTYPE      = CL.CTYPE          -- 1-физ, 2-юр. лицо
                and ATB.RESIDENT   = CL.RESIDENT       -- R-резидент, N-нерезидент
                and ATB.PRODUCT_ID = CND.PRODUCT_ID    -- Продукт
                and ATB.CURR       = AC.CURR           -- Валюта
                and ATB.TERM_OVER  = NVL(AC.TERM_OVER, 1) -- 1 - несрочный овердрафт, 2 - срочный
                and TO_DATE(SYS_CONTEXT('DWH_USER', 'DWH$START_DATE'),'dd.mm.yyyy') between ATB.DWH$START_DATE and ATB.DWH$END_DATE
                AND ATB.dwh$row_status = 'A'
              inner join S02_KONTS_ACCNT_T_B_ACCNT ATBA
               on ATBA.ACCNT_TPL_BUH_ID = ATB.ACCNT_TPL_BUH_ID
```

```
        and TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between ATBA.DWH$START_DATE
and ATBA.DWH$END_DATE
        and ATBA.DWH$ROW_STATUS = 'A'
        where CND.DWH$LOAD_DATE
=TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy')
        and  CND.DWH$CHANGE_TYPE='I';

    commit;

    insert /*+APPEND*/into ETL$_S02_DACCL_GID (DACCL_GID1,
DACCL_GID2, DACCL_GID3, DACCL_GID4)
    select /*+ ORDERED */
    X.CARD_ACCT
    ,X.CURR
    ,X.CARD_KIND
    ,ATBA.DIC_ACCNT_BUH_TYPE_ID
    from  S02_KONTS_ACCNT_TPL_BUH_CH ATB
    inner join (
    select
AC.CARD_ACCT,AC.CURR,AC.CARD_KIND,CL.BRANCH,CL.CTYPE,CL.R
ESIDENT,CND.PRODUCT_ID,AC.TERM_OVER
    from S02_KONTS_ACCNT AC
    inner join S02_KONTS_CLIENT CL
    on  CL.CLIENT    = AC.CLIENT
    and CL.CARD_KIND = AC.CARD_KIND
    and TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between CL.DWH$START_DATE and
CL.DWH$END_DATE
    and CL.DWH$ROW_STATUS = 'A'

    inner join S02_KONTS_CONDIT CND
```

```
        on  CND.CARD_KIND = AC.CARD_KIND
        and CND.COND_SET  = AC.COND_SET
        and CND.CURR      = AC.CURR
        and                          TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between CND.DWH$START_DATE and
CND.DWH$END_DATE
        and CND.DWH$ROW_STATUS = 'A'
        where (AC.STATUS = 0 or AC.END_BAL<>0 or AC.DEBIT<>0 or
AC.CREDIT<>0)
        and                          TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between AC.DWH$START_DATE and
AC.DWH$END_DATE
        AND AC.dwh$row_status = 'A') X
              on   ATB.BRANCH      = (case when X.BRANCH between 64080
and 64101 then 64001 else X.BRANCH end) --Отделение учета в ОДБ
              and ATB.CTYPE      = X.CTYPE          -- 1-физ, 2-юр. лицо
              and ATB.RESIDENT   = X.RESIDENT       -- R-резидент, N-
нерезидент
              and ATB.PRODUCT_ID = X.PRODUCT_ID     -- Продукт
              and ATB.CURR       = X.CURR           -- Валюта
              and ATB.TERM_OVER  = NVL(X.TERM_OVER, 1) -- 1 -
несрочный овердрафт, 2 - срочный
            inner join S02_KONTS_ACCNT_T_B_ACCNT ATBA
             on ATBA.ACCNT_TPL_BUH_ID = ATB.ACCNT_TPL_BUH_ID
        and                          TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between ATBA.DWH$START_DATE
and ATBA.DWH$END_DATE
        and ATBA.DWH$ROW_STATUS = 'A'
        where                                ATB.DWH$LOAD_DATE
=TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy')
        and  ATB.DWH$CHANGE_TYPE='I';
```

```
commit;

insert /*+APPEND*/into ETL$_S02_DACCL_GID (DACCL_GID1, DACCL_GID2, DACCL_GID3, DACCL_GID4)
select /*+ ORDERED */
X.CARD_ACCT
,X.CURR
,X.CARD_KIND
,ATBA.DIC_ACCNT_BUH_TYPE_ID
from  S02_KONTS_ACCNT_T_B_ACCNT_CH ATBA
inner join S02_KONTS_ACCNT_TPL_BUH ATB
on ATBA.ACCNT_TPL_BUH_ID = ATB.ACCNT_TPL_BUH_ID
and TO_DATE(SYS_CONTEXT('DWH_USER', 'DWH$START_DATE'),'dd.mm.yyyy') between ATB.DWH$START_DATE and ATB.DWH$END_DATE
and ATB.DWH$ROW_STATUS = 'A'
inner join S02_KONTS_ACCNT_TPL_BUH ATB
on ATBA.ACCNT_TPL_BUH_ID = ATB.ACCNT_TPL_BUH_ID
and TO_DATE(SYS_CONTEXT('DWH_USER', 'DWH$START_DATE'),'dd.mm.yyyy') between ATB.DWH$START_DATE and ATB.DWH$END_DATE
and ATB.DWH$ROW_STATUS = 'A'
inner join (
select
AC.CARD_ACCT,AC.CURR,AC.CARD_KIND,CL.BRANCH,CL.CTYPE,CL.RESIDENT,CND.PRODUCT_ID,AC.TERM_OVER
from S02_KONTS_ACCNT AC
inner join S02_KONTS_CLIENT CL
on  CL.CLIENT    = AC.CLIENT
and CL.CARD_KIND = AC.CARD_KIND
and TO_DATE(SYS_CONTEXT('DWH_USER', 'DWH$START_DATE'),'dd.mm.yyyy') between CL.DWH$START_DATE and CL.DWH$END_DATE
```

```
        and CL.DWH$ROW_STATUS = 'A'
        inner join S02_KONTS_CONDIT CND
        on  CND.CARD_KIND = AC.CARD_KIND
        and CND.COND_SET  = AC.COND_SET
        and CND.CURR      = AC.CURR
        and                          TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between CND.DWH$START_DATE and
CND.DWH$END_DATE
        and CND.DWH$ROW_STATUS = 'A'
        where (AC.STATUS = 0 or AC.END_BAL<>0 or AC.DEBIT<>0 or
AC.CREDIT<>0)
        and                          TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy') between AC.DWH$START_DATE and
AC.DWH$END_DATE
        AND AC.dwh$row_status = 'A') X
                on   ATB.BRANCH      = (case when X.BRANCH between 64080
and 64101 then 64001 else X.BRANCH end) --Отделение учета в ОДБ
                and ATB.CTYPE      = X.CTYPE          -- 1-физ, 2-юр. лицо
                and ATB.RESIDENT   = X.RESIDENT       -- R-резидент, N-
нерезидент
                and ATB.PRODUCT_ID = X.PRODUCT_ID     -- Продукт
                and ATB.CURR       = X.CURR           -- Валюта
                and  ATB.TERM_OVER    = NVL(X.TERM_OVER, 1) -- 1 -
несрочный овердрафт, 2 - срочный
          where                                ATBA.DWH$LOAD_DATE
=TO_DATE(SYS_CONTEXT('DWH_USER',
'DWH$START_DATE'),'dd.mm.yyyy')
           and  ATBA.DWH$CHANGE_TYPE='I';
         COMMIT;
         -- собираем статистику
         Gather_Table_Stats(c_table_name);
        end ETL_S02_REF_DEAL_ACCOUNT_LINK;
```

Додаток Б

Приклад завантаження Knowledge module (LKM + IKM)

1.

```
create table <%=odiRef.getTable("L", "COLL_NAME", "A")%>
(   <%=odiRef.getColList("", "[COL_NAME]\t[DEST_WRI_DT] NULL", ",\n\t", "","")%>,
DWH$SRC_HASH_ROW VARCHAR2(4000 char),
DWH$CHANGE_TYPE CHAR(1),
DWH$TIMESTAMP TIMESTAMP ) NOLOGGING
```

2.

```
select * from
( SELECT <%=odiRef.getColList("", "[COL_NAME]", ",\n", ",", "")%>
ROW_NUMBER () OVER (PARTITION BY <%=odiRef.getColList("","[COL_NAME]", ", ", "", "UK")%>
ORDER BY <%=odiRef.getColList("", "[COL_NAME]", ", ", "", "UK")%> DESC) RNUM
from<%=odiRef.getFrom()%>
where(1=1)
<%=odiRef.getFilter()%>
<%=odiRef.getJoin()%>
<%=odiRef.getJrnFilter()%>
<%=odiRef.getGrpBy()%>
 <%=odiRef.getHaving()%>
 order by <%=odiRef.getColList("", "[COL_NAME]", ", ", "", "UK")%> DESC)
WHERE RNUM=1
select <%=odiRef.getPop("DISTINCT_ROWS")%>
<%=odiRef.getColList("", "[EXPRESSION]\t[ALIAS_SEP] [COL_NAME]", ",\n\t", "", "")%>
 from <%=odiRef.getFrom()%>
where (1=1)
<%=odiRef.getFilter()%>
<%=odiRef.getJoin()%>
<%=odiRef.getJrnFilter()%>
```

<%=odiRef.getGrpBy()%>
<%=odiRef.getHaving()%>

3.

```
INSERT INTO <%=odiRef.getTable("L", "INT_NAME", "W")%>
(     <%=odiRef.getColList("", "[COL_NAME]", ",\n          ", "","")%>,
DWH$SRC_HASH_ROW,
 DWH$CHANGE_TYPE,
 DWH$ID )
SELECT  <%=odiRef.getColList("",  "[COL_NAME]",  ",\n          ",  ","","")%>
DWH$SRC_HASH_ROW,
 DWH$CHANGE_TYPE,
<%=odiRef.getTable("L", "TARG_NAME", "A")%>_SEQ.NEXTVAL
 FROM
(SELECT <%=odiRef.getColList("", "CASE
WHEN "+odiRef.getColList("", "C.[COL_NAME] IS NULL", " AND ",
"","UK")+" THEN T.[COL_NAME] ELSE C.[COL_NAME END
[COL_NAME]",",\n","","","")%>
NVL(C.DWH$SRC_HASH_ROW,T.DWH$SRC_HASH_ROW)
DWH$SRC_HASH_ROW,
 CASE WHEN <%=odiRef.getColList("", "T.[COL_NAME] IS NULL", " AND ",
"","UK")%> THEN 'I' --gka1
  WHEN <%=odiRef.getColList("", "C.[COL_NAME] IS NULL", " AND ",
"","UK")%>AND T.DWH$ROW_STATUS='A' THEN 'D'
 WHEN <%=odiRef.getColList("", "T.[COL_NAME] IS NOT NULL", " AND ",
"","UK")%>          AND <%=odiRef.getColList("", "C.[COL_NAME] IS NOT
NULL", "AND", "","UK")%> --gka1 AND T.DWH$ROW_STATUS = 'D' --gka1
THEN 'R'
 WHEN <%=odiRef.getColList("", "T.[COL_NAME] IS NOT NULL", " AND ",
"","UK")%>          AND T.DWH$ROW_STATUS = 'A' --gka1          AND
<%=odiRef.getColList("", "C.[COL_NAME] IS NOT NULL", " AND ",
"","UK")%> --gka1 AND ( T.DWH$SRC_HASH_ROW IS NULL OR
T.DWH$SRC_HASH_ROW <> C.DWH$SRC_HASH_ROW) --gka2 THEN 'U'
 ELSE 'N' END DWH$CHANGE_TYPE
```

```
  FROM ( SELECT <%=odiRef.getColList("", "[COL_NAME]", ",\n            ", ",",
"")%>
DWH$SRC_HASH_ROW,
DWH$ROW_STATUS
FROM <%=odiRef.getTable("L", "TARG_NAME", "A")%>
WHERE to_date(:DWH$START_DATE, 'DD.MM.YYYY') BETWEEN
DWH$START_DATE AND DWH$END_DATE) T
   FULL OUTER JOIN
  ( SELECT <%=odiRef.getColList("", "[COL_NAME]", ",\n            ", ",", "")%>
DWH$SRC_HASH_ROW
 FROM ( SELECT <%=odiRef.getColList("", "[COL_NAME]", ",\n", ",", "")%>
DWH$SRC_HASH_ROW,
ROW_NUMBER () OVER (PARTITION BY <%=odiRef.getColList("",
"[COL_NAME]", ",","","UK")%>
 ORDER BY  <%=odiRef.getColList("", "[COL_NAME]", ", ", "", "UK")%>
DESC) RNUM
 FROM <%=odiRef.getTable("L", "COLL_NAME", "A")%> )WHERE RNUM=1
)C
  ON <%=odiRef.getColList("", "T.[COL_NAME]=C.[COL_NAME]", " AND ",
"", "UK")%>)
WHERE (DWH$CHANGE_TYPE IN ('I','U','D','R'))
```

Додаток В

Копії графічних креслень